12-2009

Statistical Analysis of Linear Analog Circuits Using Gaussian Message Passing in Factor Graphs

Miti Phadnis
Utah State University

STATISTICAL ANALYSIS OF LINEAR ANALOG CIRCUITS USING GAUSSIAN MESSAGE PASSING IN FACTOR GRAPHS

by

Miti Phadnis

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Engineering

Approved:

Dr. Brandon K. Eames Major Professor	Dr. Chris Winstead Committee Member
Prof. Paul Israelsen Committee Member	Dr. Byron R. Burnham Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2009

Abstract

Statistical Analysis of Linear Analog Circuits Using Gaussian Message Passing in Factor Graphs

by

Miti Phadnis, Master of Science

Utah State University, 2009

Major Professor: Dr. Brandon K. Eames
Department: Electrical and Computer Engineering

This thesis introduces a novel application of factor graphs to the domain of analog circuits. It proposes a technique of leveraging factor graphs for performing statistical yield analysis of analog circuits that is much faster than the standard Monte Carlo/Simulation Program With Integrated Circuit Emphasis (SPICE) simulation techniques. We have designed a tool chain to model an analog circuit and its corresponding factor graph and then use a Gaussian message passing approach along the edges of the graph for yield calculation. The tool is also capable of estimating unknown parameters of the circuit given known output statistics through backward message propagation in the factor graph. The tool builds upon the concept of domain-specific modeling leveraged for modeling and interpreting different kinds of analog circuits. Generic Modeling Environment (GME) is used to design modeling environment for analog circuits. It is a configurable tool set that supports creation of domain-specific design environments for different applications. This research has developed a generalized methodology that could be applied towards design automation of different kinds of analog circuits, both linear and nonlinear. The tool has been successfully used to model linear amplifier circuits and a nonlinear Metal Oxide Semiconductor Field Effect Transistor (MOSFET) circuit. The results obtained by Monte Carlo simulations

performed on these circuits are used as a reference in the project to compare against the tool's results. The tool is tested for its efficiency in terms of time and accuracy against the standard results.

(104 pages)

To my loving family and friends....

Acknowledgments

This research project has been an enriching experience right from the beginning until its completion. I could not have even dreamed of pursuing my goal this far without the enduring and inspirational guidance of my mentor, Dr. Brandon Eames. He has been a constant source of motivation throughout my course of study while pursuing a master's degree at USU. I have learned tremendously from the courses taught by him that laid a strong foundation for my accomplishment in this research project. He has provided me guidance and support at every step to help me clear all the hurdles encountered on my way. He has always inspired me to put in my best of efforts in order to achieve my goals. I am really grateful to him and would like to sincerely thank him for all his guidance and precious time that he has devoted for this project.

I would also like to thank my committee members, Dr. Chris Winstead and Prof. Paul Israelson, for their kind support and motivation. I would especially like to extend my gratitude towards Dr. Chris Winstead for being an active part of this research. His valuable ideas proved instrumental in paving my path towards the success of this project.

Another name that is worth mentioning here is Mr. Jeremy Goldin for reviewing my thesis and improving its grammar tremendously.

My sincere thanks to my friends Vidisha, Akshata, Gautam, Sravanthi, and Arti in Logan who have been a tremendous emotional support for me in my tough times. I have shared happy, as well as sad, moments with all of them and would always cherish those memories in my heart. I would especially like to mention Sravanthi who has also been a colleague and fellowmate in several of my classes. She has helped me immensely in improving my coding and debugging skills and guided me whenever I was stuck with some problem. She also reviewed my thesis and helped me improve it at all levels.

And last, but by no means least, I am highly grateful to my family for being the greatest strength behind all my success. I would like to thank my father, Mr. Arun Phadnis, my mother, Mrs. Swati Phadnis, and my brother, Mr. Vishal Phadnis, for making me explore

my strengths and apply them towards betterment of my life. Without their blessings, I could not have become the individual that I am today.

Miti Phadnis

Contents

List of Tables

List of Figures

Chapter 1
Introduction to the Factor Graph Technique for Modeling Analog Circuits

1.1 Thesis Objective

The impact of statistical parameter variation on yield is an issue of interest in integrated analog circuit design. Due to the inherent challenges in the constituent processes, device fabrication introduces random perturbations to circuit parameter values, some of which can highly affect the performance of a circuit. The performance of analog integrated and mixed signal circuits heavily depends on the electrical parameters of their components. Often, two or more devices in an analog integrated circuit require to have identical characteristics or parameters, however device fabrication techniques introduce random deviations in the electrical parameters of the components leading to a mismatch in the values and, thus deteriorating the performance of the circuit. The performance degradation can, sometimes, be severe, rendering the circuit defective. Several random processes are involved in the production of analog circuits at different stages of fabrication that make the parameter values in a particular manufactured instance of a design uncertain, deviating randomly from their nominal values. Extreme operating conditions are also responsible for inducing such variations in the parameter values. Noise is another significant factor that causes variability affecting the operation of a circuit. For example, mismatch in the devices has been found to have an adverse impact on the digital logic schemes in several memory systems where uncertainties get introduced in the delay times, thereby causing race conditions to occur [1]. As the size of circuit components decrease, these effects become even more prominent. With the semiconductor industry advancing towards submicron era, it is becoming increasingly important to evaluate the tolerance of circuit performance on the parameter fluctuations

for reliability [2]. Significant research has been done in this area of statistical yield analysis for analog circuits [2–7]. Most of the techniques have analyzed the impact of parameter variation on yield producing results comparable with standard Monte-Carlo/Simulation Program With Integrated Circuit Emphasis (SPICE)-based simulation methods in terms of accuracy while utilizing much less Central Processing Unit (CPU) time [2, 4, 5].

Monte Carlo and SPICE have been adopted as standard simulation techniques in the analog circuit community to perform statistical yield analysis of analog circuits. These techniques perform thousands of simulations runs on several versions of a circuit instantiated to accommodate the component parameter variation range and arrive at the output statistics, thereby rendering it a very time-consuming process utilizing a lot of CPU cycles. The objective of this thesis is to introduce a novel factor graph-based statistical yield analysis method that would serve as an alternative to the already adopted techniques in the market for obtaining yield estimates for a circuit for known parameter variations. It is aimed to provide similar accuracy comparable to Monte Carlo/SPICE techniques while utilizing much less CPU time. The advantage of the proposed technique lies in its efficiency in terms of execution time at the cost of tolerable sacrifice in the accuracy of the desired results. The tool is capable of solving the traditional problem of estimating the effect of parameter variations on the yield. However, the uniqueness of the strategy is attributed to its capability of estimating unknown parameter variation tolerances for known yield statistics. It is this property of the factor graph based analysis that stands it out among other available techniques producing results comparable to the standard approaches.

Our analysis technique is built on the concepts of the factor graph theory that provides the basis for obtaining unknown yield and parameter estimates. The outcome of our research is an end-to-end design tool chain that supports the modeling of analog circuits and factor graphs, as well as the technique for the analysis. Factor Graph models are obtained by applying a translation algorithm to the circuit models specified by the user, which in turn are fed to the factor graph simulator to perform the desired analysis. The modeling environments for analog circuits and factor graphs are designed using Generic Modeling En-

vironment (GME). GME has been the fundamental element in the development of the tool chain and forms the backbone of our tool structure. We have discussed the background information on GME in Chapter 2. Section 1.2 introduces factor graphs and their significance in our approach.

1.2 Factor Graphs

Graphical models, like signal flow graphs, trellis diagrams, and a variety of block diagrams, have often been utilized by engineers to model their systems [8]. Factor graph analysis is another emerging technique in modeling theory and has been used in a variety of domains. It has the ability to model various types of control and signal processing systems. A factor graph is defined as a bipartite graph capable of realizing a global mathematical function as a composition of the factors of several local functions [9] and provides a graphical representation of such factorization. There exists a unique node for every factor and a unique edge for every variable in the graph; a factor node is connected with an edge representing some variable only if that factor is a function of that variable [10]. Edges connecting nodes in the graph model the factorization dependencies of the global function. Factor graphs are associated with a summary propagation algorithm that operates by passing messages along the edges of the graph [8]. Several algorithms have been devised as an instance of this summary propagation algorithm to analyze the factor graphs [11]. It thus allows one to derive the transfer function of the system represented by the graph through evaluation of the output for a given set of inputs by message propagation.

Factor graph theory has been applied to the analysis of mixed signal circuits. Loeliger [12] has explored the potential of the message passing algorithm in a factor graph for the calibration of analog to digital converters made of low precision components. However, the work performed to date has examined only a very narrow domain of the application of factor graphs and the associated sum-product algorithm in the area of analog circuits. In this thesis, we have attempted to widen the scope of the factor graph theory so as to generalize the design automation process for different kinds of analog circuit components and circuits, performing steady state yield analysis. Our focus is to provide the ability to model

all kinds of linear and nonlinear circuit elements such as amplifiers and Metal Oxide Semiconductor Field Effect Transistor (MOSFET), and to apply factor graph analysis to obtain time efficient simulation results with appreciable accuracy. The intention is to explore the potential of the approach in both directions; to be able to solve the traditional problem of determining yield estimates for known parameter variations; and to perform the unconventional unknown parameter estimate analysis for known yield statistics, which could be thought of as a kind of sensitivity analysis. Factor graphs can lead to the formulation of an efficient technique, in terms of accuracy and execution time, for performing statistical yield analysis of analog circuits in the presence of random parameter variations in circuit components, as well as evaluating the constraints on unknown parameter estimates for known yield statistics.

1.3 Thesis Organization

The thesis has been organized into the following chapters:

- Related Background and Literature Review: This chapter delves into the background details of two crucial components involved in the development of the technique: Factor Graphs and GME. It also discusses different statistical yield analysis techniques and various application domains of factor graphs.

- Technique of Leveraging Factor Graphs for Statistical Yield Analysis of Analog Circuits: This chapter gives a detailed explanation of the proposed analysis technique and the tool with the help of a few circuit examples.

- Simulation Results Obtained by the Tool: This chapter demonstrates experimental results of the tool for three different operational amplifier circuits and a MOSFET circuit, and also performs a comparative study between the proposed approach and the standard Monte Carlo simulation.

- Conclusion and Future Work: The last chapter provides a concluding note on the proposed technique and also discusses future work.

Chapter 2

Related Background and Literature Review

Section 2.1 briefly introduces the two major tools involved in the development of the factor graph technique proposed in this thesis.

2.1 Related Background

Factor Graphs and GME are the major contributors towards realizing the tool chain that performs factor graph-based analysis of analog circuits. They form the crux of the proposed approach and its implementation. A brief discussion of factor graphs and GME is presented in secs. 2.1.1 and 2.1.2.

2.1.1 Factor Graphs

Factor graphs have gained significant recognition in the engineering community as powerful system modeling tool. As mentioned in Chapter 1, they have been widely explored in various domains including signal processing and control systems. A factor graph models the factor dependencies in the transfer function of a system to represent the system output in terms of user specified inputs. It performs quantitative analysis of a physical system based on the mathematics involved in the structural components of the graph. A factor graph resembles older modeling techniques like Tanner graphs and Bayesian networks. Factor graphs and the sum-product algorithm are considered an easier approach with close correspondence to Tanner graphs for solving marginalized product-of functions problem [11]. A factor graph expressing certain factorization is closely related to a Bayesian network representing the same factorization. There exists functional similarity between the belief propagation algorithm that operates by passing messages in a Bayesian network and the sum-product algorithm passing messages in a factor graph. A generic message passing algo-

rithm known as sum-product algorithm that operates on a factor graph has been discussed by Kschischang, Frey, and Loeliger [11]. This algorithm computes the local factor functions associated with the global function represented by the graph [11].

The basic building blocks of a factor graph include mathematical entities like adders, multipliers, equality constraint nodes, coefficient and error source blocks, as well as input-output nodes representing the inputs and outputs of a system. Factor graph theory defines equations for calculating means and variances of Gaussian messages that are propagated to each block in the factor graph model. These equations have been established by Loeliger [10] with a few exceptions discussed later. Equations are also defined for analyzing Gaussian mixtures. These equations guide the propagation of signals along all forward paths of the graph starting with the known inputs. Backward propagation is also defined, which can be used to arrive at an estimation of an input parameter using known output statistics. Equations associated with few factor graph nodes that guide the signal propagation in both, forward and backward directions are described below:

- Adder: This node is responsible for adding or subtracting (depends on the signal polarity) the two incoming Gaussian input signals. It is defined by the equation

$$Z = X + Y. \tag{2.1}$$

  The values of Mean(M), Variance(V) and Weight(W) associated with the Gaussian output message obtained by the forward propagation and Gaussian input messages obtained by the backward propagation are given below:

  1. Forward Propagation

$$Z.M = X.M + Y.M \tag{2.2}$$

$$Z.V = X.V + Y.V \tag{2.3}$$

$$Z.W = 1/Z.V. \tag{2.4}$$

2. Backward Propagation

   Outgoing Message on X:

$$X.M = Z.M - Y.M \tag{2.5}$$

$$X.V = Z.V + Y.V \tag{2.6}$$

$$X.W = 1/X.V \tag{2.7}$$

   Outgoing Message on Y:

$$Y.M = Z.M - X.M \tag{2.8}$$

$$Y.V = Z.V + X.V \tag{2.9}$$

$$Y.W = 1/Y.V. \tag{2.10}$$

- Multiplier: The Multiplier takes the product of the input Gaussian signal with a multiplication factor attribute to produce an output. The factor can be a scalar, a vector, or a matrix. It is represented by the equation

$$Z = A * X. \tag{2.11}$$

  X corresponds to the input signal, A is the multiplication factor, and Y is the output. The equations illustrating the calculation of forward and backward Gaussian messages through this node are as follows:

1. Forward Propagation

$$Y.M = A * X.M \tag{2.12}$$

$$Y.V = (A^2) * X.V \tag{2.13}$$

$$Y.W = 1/Y.V. \tag{2.14}$$

2. Backward Propagation

$$X.W = Y.W * A^2 \tag{2.15}$$

$$X.M = A * Y.W * Y.M/X.W \tag{2.16}$$

$$X.V = 1/X.W. \tag{2.17}$$

An example factor graph consisting of basic factor graph blocks is shown in fig. 2.1. In this example, epsilon represents an error source block, C denotes the coefficient node, I/P1 and I/P2 correspond to the input nodes of the graph, and O/P represents its output.

2.1.2 GME

GME stands for "Generic Modeling Environment." It is an academic tool developed at Vanderbilt University for the purpose of defining modeling languages specific to a domain. GME is a configurable toolset that facilitates easy creation of domain-specific modeling environments [13]. The configuration is accomplished through metamodels specifying the modeling paradigm for the application domain [14]. A paradigm refers to the domain language that embeds all the constructs for modeling an application within it. It contains all the information concerning syntax and semantics of the resultant modeling environment. GME allows to customize modeling paradigms for various application domains. It also provides inherent support for a handful of paradigms such as UML and MetaGME. UML defines a graphical user interface to create UML class diagrams. MetaGME is a special meta modeling paradigm of GME which facilitates the creation of a metamodel, defining a modeling language for a particular domain. It reflects the meta programmable capability

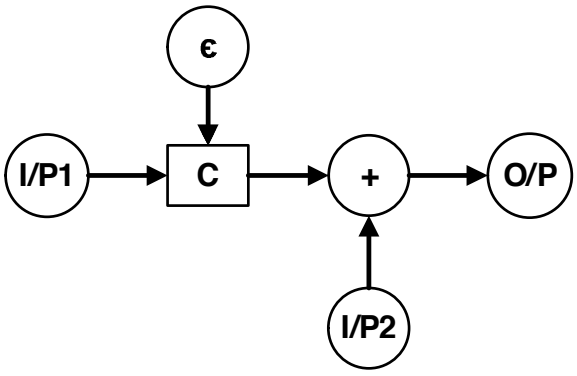

Fig. 2.1: Example of a factor graph.

of GME enabling to generate domain-specific environments that could be used to create domain models. It is even possible to specify the meta-meta model dictating the syntax and semantics of a meta model defining a paradigm.

The metamodel provides an impression of UML class diagram representing a hierarchical structure with a root and several branches. The structure is composed of GME's basic building blocks required to specify the design environment. These blocks represent generic concepts implemented by GME that are abstract enough to be applicable to a wide range of domains. They include models, atoms, and connections. A model entity in the metamodel specifies a class data structure that is capable of containing other entities within it. It encapsulates all the information about that object's parents, children, and attributes. On the contrary, atoms are the fundamental entities of a metamodel contained in a model which could not be divided further into constituent parts. Connection objects are used to express relationship between two objects contained in a model. These entities form the structural basis of the models conforming to the modeling paradigm defined by a metamodel.

The MetaGME interpreter tool of GME facilitates the creation of the modeling paradigm from a metamodel. It extracts all the syntactic information specified by the metamodel and embeds it into a configuration file. This file contains multiple API's to gather model-specific information. It also allows to define interpreters corresponding to each custom defined modeling paradigm. An interpreter is a user-developed software program that utilizes the API's specified by the paradigm to perform structural and semantic analysis of generated models. It could be dedicated to perform a specific task on the model as per the requirements of the application. GME supports creation of interpreters in multiple languages including C++, Visual Basic, Python, etc. [14].

2.2 Literature Review

This section discusses various simulation and yield analysis techniques adopted targeting the domain of analog circuit design. It also highlights the significance of factor graphs as a powerful modeling and analysis tool by describing the wide range of applications to which it has been applied.

2.2.1 Overview of Different Analog Circuit Simulation Techniques

The domain of statistical analysis of yield has captured much attention in the past. Many techniques have been proposed to perform this analysis which are faster as compared to the standard Monte Carlo/SPICE simulation techniques.

Variance analysis has emerged as one of the strategies to estimate the effects of random device parameter variation on analog integrated circuits [4]. It is particularly effective when device variations can be accurately modeled as random variables with Gaussian distributions, and where each such variable is statistically independent. The proposed analysis involves two steps.

- Linear Variance Calculation- The function of random parameters describing the output is weakly nonlinear, and hence is approximately linearized. In this approximation, stochastic influences of all random parameters are modeled using a single random variable that randomly deviates from the nominal value of the yield. The random variable and the yield, both posses a normal distribution.

- Computing the influence of nonlinearities- The second order sensitivity coefficients account for the nonlinearity of the function of random parameters due to which yield is no longer normally distributed.

Many analog circuits are weakly nonlinear within their parameter variation range, implying negligible effects of second order sensitivity coefficients on the output mean and variance. This has been proved by considering two practical examples of a transconductance amplifier and analog filter [4]. Taking Monte Carlo simulation as the reference point, the results of the variance-calculation method match well for both example circuits. Moreover, the linear approximation results vary only slightly from those obtained with second order sensitivities. The variance calculation method has proved better than Monte Carlo analysis in terms of computational efficiency for predicting circuit characteristics with parameter dependencies with only some small sacrifice in accuracy.

Symbolic analysis is another formal approach developed to evaluate the behavior or a specific characteristic of linear circuits in which a few or all circuit elements are represented

as symbols [7]. Software is used to translate a circuit description into a symbolic expression modeling the circuit characteristic of interest. This analysis is primarily restricted to linear circuits. Algebraic (matrix or determinant-based) and graph based methods are two basic classes of symbolic analysis methods used to analyze the generated symbolic expression. As the symbolic expression obtained by this method is usually very complicated, it could be further preprocessed to obtain a less complex and more comprehensible solution which has only the dominant contributions. Conventionally, graph-based methods have been considered most suitable for obtaining fully symbolic network functions in the past. The target application domain of symbolic circuit analysis includes analog integrated circuits. It plays an important role in the automation process of analog circuit design, as it gives insight into the behavior and trade offs of analog circuits, and also helps in circuit sizing and testability analysis. It also bears potential for the development of analog CAD tools. The applications of symbolic analysis area are as follows:

- Insight into Circuit Behavior- Symbolic simulator generates correct analytic expressions in a much shorter time and for more complex characteristics and circuits which remain valid even when the numerical parameter values change. It is also useful for the designers to obtain analytic expressions for second order characteristics. However, it is limited by the circuit size and the type of analysis.

- Analytic model generation for automated circuit sizing- It can automatically generate all AC characteristics in the analytic model of a circuit which is then used to size the circuit in an optimization program.

- Iterative circuit (design space) exploration and topology generation- It is used to iteratively explore and improve new circuit topologies. Influence of topology changes do get reflected in the analytic expressions.

- Repetitive formula evaluation- In this case, symbolic expressions for the network functions are only compiled once and then can be evaluated multiple times for particular values of the circuit and input parameters. Its obvious applications include statistical

analysis (Monte Carlo simulations), large-signal sensitivity analysis, yield estimation, and fault diagnosis of linear and nonlinear analog circuits.

Some of the successful modern simulators include ISAAC, ASAP, SYNAP, SSPICE, etc. These are targeted towards symbolic analysis of analog integrated circuits with a built-in small signal linearization and symbolic approximation of the expressions. Unsolved areas in this field constitute symbolic analysis of large-signal behavior, time domain behavior, and strongly nonlinear circuits. There also lies potential for the improvement of post processing capabilities to improve the interpretability of the generated expressions.

There have been efforts to design a simulation-based technique that includes yield as one of the performance parameters in the optimization process. Ali et al. [6] have proposed a predictive model to include yield as the cost function in the initial stages of the design and then use Monte Carlo simulation for its estimation. The two main components of the synthesis model include Optimizer and Evaluator. The Optimizer finds the best suited parameters of the circuit that meet all the specifications and provide higher yield. This circuit specification with all the parameters is fed to the Evaluator which determines the fitness score and feeds it back to the Optimizer. The algorithm used by the optimizer to search best possible solution is the global stochastic search algorithm called Genetic algorithm (GA) technique. In the GA technique, any one of the possible topologies of an analog circuit that meets specifications is randomly chosen for which a SPICE netlist is created and desired parameters are generated. These parameters are then parsed to a netlist and HSPICE simulation is performed on them. The spice result is parsed to the GA and ranked. The process is stopped when a specified number of generations are reached and the best solution is chosen according to the rank, otherwise a new generation is created to iterate the above procedure. This method is slow because Monte Carlo simulations consume a lot of CPU time. These ideas have been implemented for an operational transconductance amplifier [6]. The results obtained for this circuit example are compared with the non-yield predictive approach which showed that the proposed strategy results in higher yield. However, computational cost is increased in order to achieve greater accuracy. Rodriguez-

Macias and Rodriguez-Vazquez [5] have developed a strategy for reducing the time taken to calculate mean and variance of analog cell specifications in the presence of random variations in the component parameters. The strategy involves performing AC analysis for analog circuits. The accuracy of the method is comparable to standard Monte Carlo simulations. AC analysis is typically preceded by a DC analysis of the circuit in the beginning to obtain DC operating point for linearizing the device model. Its speed depends on the number of circuit elements and the number of frequencies to be analyzed. There are two strategies proposed [5] to reduce this time. The first method reduces the time required to evaluate specifications at one frequency. The analysis of different circuit configurations requires solving a system of linear equations for each of those configurations at all the frequencies. Such calculations are usually very time consuming. The technique of "Increased Principle Matrix" solves just one system of equations with a bigger range, thus making it faster. The second strategy called "Grouping of Principal Matrices into Equivalence Classes" groups principal matrices of the samples obtained from Monte Carlo analysis with similar operating points into one entity instead of analyzing them separately.

The two approaches discussed above have been implemented in a tool called FASTEST, and its results have been found to match closely with HSPICE simulations for a large number of analog cells. The time taken by the tool to evaluate circuit characteristics is much lower than the surface response method since techniques employed by FASTEST are independent of the number of parameters undergoing random variations.

A technique to evaluate worst case response of linear analog circuits in the presence of parameter variation has been suggested by Michael W. Tian and C.-J.Richard Shi [2]. The authors note that if the yield is monotonic with respect to a particular parameter variation over the parameter space, worst-case yield analysis considers only the corners of the space in order to determine upper and lower bounds on circuit response. The main challenge in this technique is identifying worst-case parameter sets that lead to such a circuit response. Such algorithms proposed in the past have been categorized into three separate groups; Monte Carlo simulation, Interval analysis, and Sensitivity-based vertex analysis. Monte

Carlo analysis usually underestimates the results and is very time-consuming. Interval analysis, on the other hand, overestimates the results. Vertex analysis is based on the assumption that worst case circuit response corresponds to the parameter sets located at the vertices of the parameter space. General observations indicate that worst case parameter sets of monotonic parameters are located at their corner values which could be exploited to reduce the number of uncertain parameters by replacing these parameters with their corner values. The reduction helps in improving the efficiency of the Monte Carlo analysis. This paper validates the vertex analysis by proving a theorem which states that "if the circuit response is monotonic with respect to the changes in any circuit parameter at any point in the parameter space, then the worst case of the circuit response results from the parameter space vertices." Generally in most practical circuits, monotonicity is satisfied by most uncertain parameters of the circuit, but not all of them. Worst case analysis of such circuits leads to an uncertainty-reduced circuit simulation instead of a nominal solution which is close to being accurate. One of the sections of the paper focuses on calculating a sensitivity band that defines bounds of sensitivity between circuit response and parameter. It is used to determine the kind of the monotonicity of the circuit. The worst case tolerance analysis algorithm first calculates the sensitivity band of the circuit response over all uncertain parameters and then replaces the parameters with their corner values for each circuit response, depending on the type of monotonicity being satisfied. These ideas have been implemented in a prototype circuit simulator and a high-sensitive state variable filter circuit is used to show the results.

Some work has also been done in the development and use of expert systems for automated analog circuit design. El-Turky and Perry [3] discuss an automated design methodology for analog circuits that integrates formal and intuitive knowledge into one program for the design process. BLADES is a prototype design environment based on this strategy, and is capable of designing a variety of sub-circuit functional blocks and a limited class of op-amps. It is based on the expert systems strategy of stored knowledge which can be either a formal mathematical technique or an intuitive reasoning mechanism applied by

human designers. Circuit design knowledge includes both systematic procedures to design the circuits and special rules to handle special situations. BLADES uses OPS5 production system to implement this rule base. It is an expert system building tool in LISP that implements the rules in if-then format. If represents the instance of rule application and then indicates the type of action to be taken. It also undergoes a conflict resolution mechanism to choose the best applicable rule amongst all those that qualify. There are five main parts of BLADES system architecture:

- Expert system manager,
- Sub circuit design experts,
- Knowledge base,
- Design consultants,
- Test generation.

The expert system manager is the main design engine of BLADES, which decides the circuit topology. It determines the specifications for all the sub-circuit blocks into which the entire system has been partitioned based upon the global specifications given by the user. Sub circuit experts are responsible for designing each sub circuit block using design equations and the knowledge base, which is the combination of intuitive and formal knowledge. Testing is performed to ensure the correctness of the designed circuit in meeting the input specifications. Design consultants are the circuit simulator programs that aid BLADES to prove the correctness of the design. ADVICE simulator is used as a major consultant to BLADES. BLADES uses three different hardware description languages to communicate knowledge to the system and convey the requirements at each abstraction level. The authors discuss the case study of an operational amplifier design as a practical application of expert systems in analog circuit design [3]. BLADES is regarded as the first successful expert system to design analog circuits. It has strengthened the fact that artificial intelligence could be used efficiently to apply human reasoning in the design process of loosely structured domain of circuits.

2.2.2 Application Domains of Factor Graphs

Factor graph theory has gained much recognition as a modeling approach in several areas in the recent past. Signal processing applications such as estimation and detection problems have extensively exploited the potential of factor graphs. Several algorithms used in the fields of Artificial Intelligence (AI), Digital Signal Processing (DSP), and communications like the forward/backward algorithm, Viterbi algorithm, Kalman filter, etc., are instances of the basic sum-product algorithm [11]. Tabulated forms of computation rules associated with each building block of the factor graph representation of linear model have been presented [10, 15].

Loeliger et al. [15] discuss a Gaussian message passing approach in the factor graph of linear models. Message computation rules have been derived for the building blocks of the linear models that help in designing convenient and less computation-intensive algorithms. Message passing algorithms on linear Gaussian models can be derived as instances of basic sum-product algorithm for a variety of problems such as equalization, RLS adaptive filters, LPC analysis, etc. All the variables in such models are assumed to be Gaussians for which sum-product algorithm and max-product algorithm coincide. Forney style factor graphs are used to represent linear state space models in this paper. The approach has been applied to the problem of equalization of transmitting real symbols over an interfering channel with white Gaussian noise.

The combination of factor graphs and sum product algorithm has also been applied to the development of detection algorithms for ISI channels [16]. The technique aims at determining aposteriori probability distributions of the transmitted symbols in the Inter-symbol Interference (ISI) channels. It represents an ISI channel with the help of factor graph and the sum-product algorithm is applied to it to obtain a detection algorithm used for turbo equalization. The flooding schedule is adopted as the message-passing schedule in the factor graph where every iteration results in updating and passing new messages from all the nodes to their neighbors. This schedule is chosen because it is appropriate for the full parallel implementation of the detectors. The application of sum-product algorithm to

an ISI channel is accurate for a cycle-free graph and is independent of the used schedule, however it produces approximate results for a graph with cycles. Factor graphs describing channels usually have cycles leading to an iterative detection process and performance of such algorithms is close to being optimal for the graph of girth 6. The performance of the proposed technique has been accessed using the values obtained with series BCJR algorithm as the benchmark for different kinds of channels [16]. BCJR algorithm is a standard accepted technique to perform maximum aposteriori decoding of error correcting codes. It is named after its inventors: Bahl, Cocke, Jelinek, and Raviv [17]. The results of the technique match closely with the exact marginal values obtained by BCJR algorithm. The proposed algorithms provide a very high-speed detection and their complexity could be reduced more efficiently as compared to optimal detection schemes. Moreover, combined detection and decoding is possible with factor graphs due to the inherent parallel structure of sum-product algorithm unlike the BCJR algorithm.

A technique of parameter estimation in a Gaussian auto-regressive model by message propagation in a factor graph has been discussed [18]. An auto-regressive (AR) model is a type of random process that is used to model different types of natural phenomena [19]. It focuses on the joint estimation of AR parameters, noise variance, and innovation variance for known observations of the output. Parameters are estimated from the marginals of the global function obtained by passing messages in the graph. Recursive Least Squares (RLS) and Cryptographic Message Syntax (CMS) algorithms are shown as the instances of basic message passing algorithm for estimating AR parameters [18]. The propagation algorithm works fine with forward-only propagation for zero noise variance and known innovation variance. However, in the cases of non-zero unknown or known noise variances, messages have to be propagated back and forth iteratively along the edges of the graph to arrive at an AR parameter estimate.

The expectation Maximization algorithm has been shown as an instance of the message passing algorithm [20]. The sum-product algorithm is used to marginalize over the hidden random variables in the model. EM algorithm estimates unknown parameters in a system

by calculating maximum likelihood (ML) parameter estimates. It has been widely applied to the fields of communication and signal-processing. This paper discusses the technique of EM algorithm with two example systems, an LDPC code transmitted over a Rayleigh block fading channel with white noise and block binary source in an LDPC-based slepian-wolf source encoder. The results of factor graph EM algorithm for these two examples are compared with a one-stage detector and a detector that assumes memory less Rayleigh fading and appreciable gain has been observed with the FGEM algorithm.

Chapter 3

Leveraging Factor Graphs for Statistical Yield Analysis of Analog Circuits

The main objective of our technique is to develop an approach, given a linear analog circuit model, for deriving a factor graph model corresponding to the analog circuit and applying Gaussian message passing to it to perform yield analysis. An end-to-end tool chain is implemented to facilitate the translation of the circuit into a factor graph and perform statistical analysis using parameters specified by the user. The two primary objectives of this technique are:

- To obtain output signal statistics for known component statistics in an analog circuit;
- To estimate the tolerance limits on statistics of circuit components for a specified yield.

As with other factor graph analysis approaches, we assume that all statistical parameter variations are normally distributed and are independent of each other. Our current focus is the steady state analysis in the presence of circuit parameter variation by simultaneously evaluating all the circuit instantiations resulting from the fabrication process.

A factor graph consists of well-defined set of building blocks which can be used to model different physical elements of an analog circuit. There are two major groups of variables in the factor graph equivalent model of an analog circuit: voltage variables for each node in the circuit and current variables corresponding to every branch in the circuit containing a resistor. Kirchhoff's laws of voltage and current play a guiding role in mapping the series and parallel combination of circuit elements, particularly resistors, onto their corresponding combination of factor graph blocks. The connections between factor graph blocks correspond to the topology of the circuit. Gaussian signals with known mean and variance

are used to model circuit parameter variations. These messages are propagated along paths from the input nodes to output nodes of the factor graph. The graph nodes traversed along these paths represent defined equations which perform transformations on the signals. Factor graph theory defines equations for both forward and backward propagation of signals through the interior graph nodes. The established equations of graph theory discussed by Loeliger in his paper [10] are utilized to develop our analysis technique. These semantics have been leveraged in our technique for deriving mathematical expressions to quantify the behavior and performance of analog circuits.

The method proposed in this thesis involves the derivation of a formal representation of linear analog circuits in terms of factor graphs and applying factor graph theory to carry out statistical circuit analysis. Analysis is conducted to predict the behavior of a circuit in the presence of parameter variations. A useful feature of the factor graphs is their ability to support merged statistical analysis by propagating the messages in both forward and backward directions in the graph. Forward propagation is the ability to arrive at unknown output statistics starting with the known input statistics propagated in the appropriate direction in the graph. Forward propagation of messages along the edges of a factor graph model of an analog circuit suggests an alternative to the traditional method of determining the yield as a function of component variation. Conversely, backward propagation carries information known or imposed on circuit output nodes in the direction opposite to the signal flow in the circuit, refining the statistics of variables modeling circuit parameters. This backwards analysis permits the imposition of a yield constraint, in the form of restricted output variance, on a design in order to determine its impact on the statistics of circuit parameters. The level of impact provides a measure of parameter sensitivity to yield making backward propagation a form of sensitivity analysis. This research has provided the ability to apply the bi-directional nature of factor graph analysis to the determination of parameter variation's influence on yield and vice-versa in an analog circuit. Both sets of analysis runs have been performed to evaluate the tool's performance in yield calculation by forward message propagation and unknown parameter estimation by backward message propagation

in the graph.

3.1 Correspondence Between Analog Circuit and Factor Graph Domain

Yield estimation of analog circuits using factor graphs is based on the theory that analog circuit models are strongly analogous to factor graph models for the task of behavior simulation. In order to accomplish the task of translating an analog circuit into its equivalent factor graph model, we have defined a set of translation rules for each type of analog circuit element. The approach is generalized in the sense that every circuit component maps onto a single or fixed combination of factor graph blocks regardless of the type of the circuit. The rules for converting different circuit components into factor graph blocks are listed in the following subsections.

3.1.1 Independent Signal Sources

Ideal input sources of voltage and current in a circuit are represented by the input nodes of a factor graph and they serve as the starting points of signal flow along the graph. The Gaussian input signals with known mean and variance are specified by the user in the circuit model and represent the expected statistics resulting from circuit fabrication. This information is provided to the factor graph input nodes to act as the starting points for message propagation.

3.1.2 Resistors

Passive devices such as resistors translate into a fixed combination of two factor graph blocks connected together in the factor graph model, a coefficient node and an error source block. A coefficient node models the properties of a resistance in the graph, while an error source block, which serves as the input to the coefficient node, models the variation in the fabricated resistance's value. The variation in the values of a resistance due to the fabrication techniques is assumed to be linearly distributed. The translation rule for a resistance is based upon this linear approximation of resistor values in the presence of parametric variations, neglecting the higher order coefficients. The coefficient node is a

two-input node that accepts the input signal to the resistor in the analog circuit domain and the signal from the error source block as its two inputs. The outcome of the coefficient node is the scalar product of the two input signals with the resistor value. Mathematically, the combination of these two factor graph nodes that model a resistor is represented by the following equation:

$$Z = X * A * (1 + eps), \tag{3.1}$$

where X is the input signal to a resistor, Z is the output signal from it, A represents the resistor value, and *eps* denotes the error source factoring in the variance associated with the randomness of the resistance value. The signal initiated from the error source block is a normally distributed signal with zero mean. Graphically, we capture the resistor and its equivalent factor graph as depicted in fig. 3.1.

There exists a voltage subgraph and a current subgraph in the factor graph model for every resistor in the circuit. The two sub graphs have been depicted graphically in fig. 3.2. They guide the transformation of the signal propagating in the associated branch according to the electrical laws obeyed by the resistor. The two sub graphs model Ohm's law and calculate the voltage drop across the resistor. They are connected by means of an adder node and this whole combination evaluates the voltage drop across the resistance according

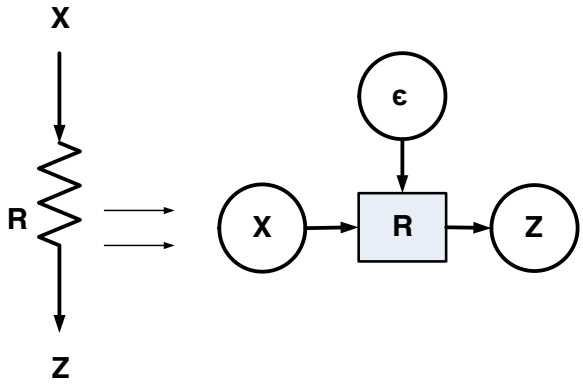

Fig. 3.1: Symbol of a resistor and its equivalent factor graph.

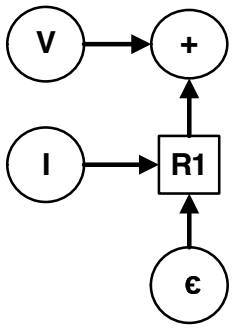

Fig. 3.2: Voltage and current subgraphs in a factor graph.

to the equation

$$V = I * R. \tag{3.2}$$

Here, I denotes the current flowing in the branch with resistance R, and where V is the input voltage at one of the ends of R. The current subgraph consists of an input current node I and the combination of coefficient and error source nodes representing the resistor while the voltage subgraph is either made of an input factor graph node or a combination of other factor graph nodes representing an input voltage V at one of the ends of the resistance.

3.1.3 Amplifiers

Amplifiers are modeled as a two-terminal device with the ability to specify inputs at both the positive and negative input terminals. If one of the terminals is grounded, a junction node with zero voltage is connected as an input to it. Amplifiers map onto a series combination of an adder and a scalar multiplier node in the factor graph. The amplifier node and its factor graph equivalent combination are shown in fig. 3.3. The two inputs to the adder represent voltages at the two terminals of the amplifier. The resulting sum is fed to the multiplier node connected to the adder. The output signal of this node is the product of the incoming signal with the specified open-loop gain of the amplifier that is specified as the gain attribute of the multiplier node. The fundamental amplifier element offered by the tool represents an ideal op-amp with a very high open-loop gain, however nonideal op

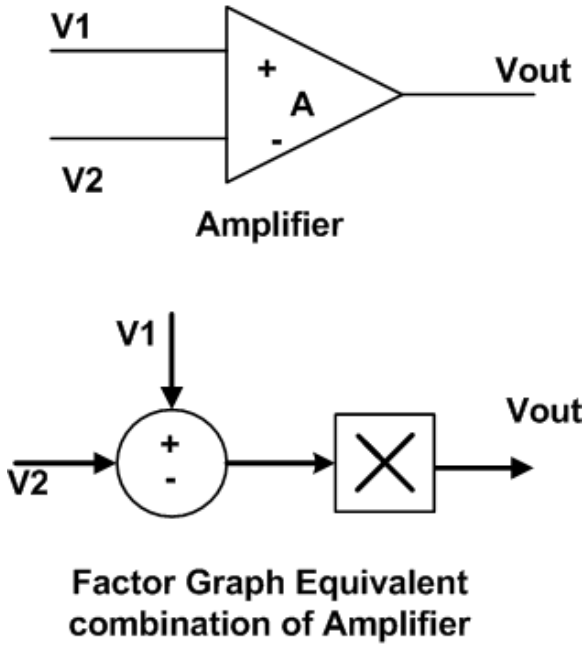

Fig. 3.3: Symbol of amplifier and its equivalent factor graph.

amps could also be modeled by the combination of an ideal amplifier and other available parts.

3.1.4 Dependent Sources

We have also devised rules to translate dependent current and voltage sources into factor graph blocks. The ability to represent dependent sources in the factor graph domain allows modeling and analysis of MOSFET circuits using factor graphs. The tool facilitates the modeling of the following four kinds of dependent sources.

1. Current Dependent Current Source: This is a dependent current source whose output current is controlled by an independent current source connected across a resistor in some branch in the circuit. The factor graph equivalent of such a source includes a scalar multiplier node. The independent current source controlling the dependent source provides input to the multiplier block that factors in the amplification factor of the dependent current source into its gain attribute. The output signal yielded by the scalar multiplier node is the amplified current signal initiated by the dependent

current source.

2. Voltage Dependent Current Source: In this case, the current output of the source is controlled by the voltage between two nodes in the circuit. This type of a dependent source maps onto the combination of an adder node connected to the scalar multiplier node in a factor graph. The voltages at the two controlling nodes serve as the inputs to the adder, the output of which is fed to the multiplier node. Its gain attribute corresponds to the conductance offered by the voltage dependent current source and yields the current initiated by it.

3. Current Dependent Voltage Source: This is similar to the current dependent current source as its output voltage is controlled by an independent current source in the circuit. It also maps onto the scalar multiplier node in a factor graph whose gain attribute represents the resistance offered by the source and produces an output voltage corresponding to the dependent voltage source.

4. Voltage Dependent Voltage Source: This source is translated into the combination of an adder block connected to the scalar multiplier node, similar to the dependent current source. The adder output represents the controlling voltage while the output of the multiplier represents the amplified output voltage produced by the dependent voltage source.

3.1.5 Junctions

Junctions in a circuit could be of two kinds: one that broadcasts the incoming signal along all available paths in the circuit, and the other that merges the signals from several branches into one. Both types of junctions are represented by the combination of a variety of factor graph nodes in the graph with at least one equality constraint node. Figure 3.4 provides two example merge junctions and their corresponding factor graph blocks. An equality constraint node distributes the incoming signals approximately equally in all directions. Equality nodes allow the distribution of current and voltage signals over multiple parallel branches or nodes in the circuit. They play an important role in modeling feedback

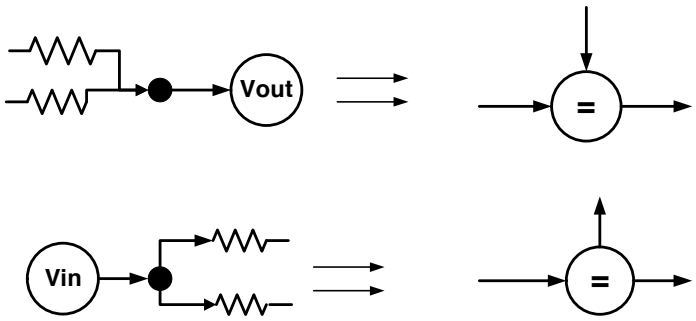

Fig. 3.4: Circuit fragments with merge junction and their corresponding factor graph blocks.

loops in a circuit and the equations associated with them suitably quantify the effects of feedback. Junctions are critical points in a circuit and exhibit different rules of translation guided by the application of Kirchhoff's laws of voltage and current at those nodes to correctly model the distribution of voltage and current signals across them. In order to be able to model most of the circuit topologies, we have defined nine different kinds of junction possibilities and implemented translation rules for them. The different types of junctions are categorized based on the number of input and output branches coming into and out of them, respectively. They range from a junction with zero inputs and two outputs to a junction with two input and two output branches. They have been devised intuitively such that they can permit the generalization of the translation algorithm, and can be applied to a variety of different analog circuits.

All related information about the type of analog elements connected at the inputs and outputs of a junction according to the circuit topology is encapsulated within a data structure. For every junction encountered in a circuit, an instance of this data structure is created. Since junctions are typically the points of voltage equivalence in a circuit, the basic equivalent factor graph structure for the junctions is comprised of an input node representing the node voltage connected to an equality constraint node. This combination results from the application of Kirchhoff's voltage law to the junction. There may or may not be an input voltage node depending on the type of the junction. This basic structure may also include a current subgraph containing an adder node with an input current node and one of the outputs of the equality constraint node as two inputs to the adder, based

upon the number of outgoing branches from the junction that contain a resistor. Merging voltage and current subgraphs facilitates the application of Ohm's law across a particular resistor connected to that junction. The equivalent factor graph models for different kinds of junctions based on the above described theory are listed below.

- Type 0: It is a junction with zero input branches and three output branches. An example circuit showing junction of this type is shown in fig. 3.5. Its corresponding factor graph block is shown in fig. 3.6. In case, if one of the output branches of this junction contains a signal source, its equivalent factor graph combination reduces to an input factor graph node connected to an equality constraint node as shown in fig. 3.7. The equivalent factor graph of the example circuit showing junction of Type 4 is shown in fig. 3.8.

- Type 1: It is a junction with zero input edges and one output edge, represented by an input factor graph node defining the node voltage in the factor graph domain. It may be connected to the current subgraph by means of an adder incase the outgoing branch from it contains a resistor. An example circuit showing junction of this type and its equivalent factor graph are shown in figs. 3.9 and 3.10, respectively.

- Type 2: This type of junction possesses one input edge and zero output edges. It translates into an equivalent output factor graph node representing that junction's voltage in the factor graph domain. An example circuit showing junction of this type and its equivalent factor graph are shown in figs. 3.11 and 3.12, respectively.

- Type 3: It has two incoming and zero outgoing branches. An example circuit showing junction of this type and its equivalent factor graph are shown in figs. 3.13 and 3.14, respectively. It converts into the combination of factor graph blocks shown in fig. 3.15.

- Type 4: It represents a junction with one input and one output edge. It acts as a channel directly connecting its input and output analog elements and does not require any conversion in the factor graph domain. However, if the input analog element connected to it is an independent current source, it transforms into an input factor

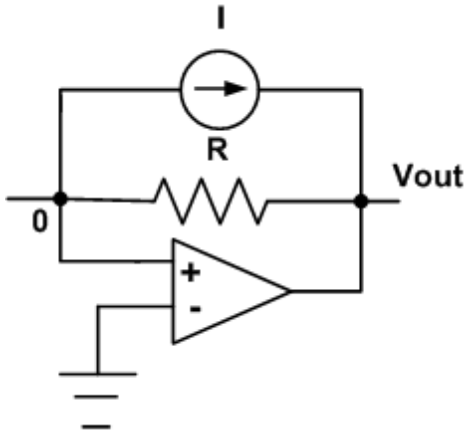

Fig. 3.5: Example circuit showing junction of Type 0.

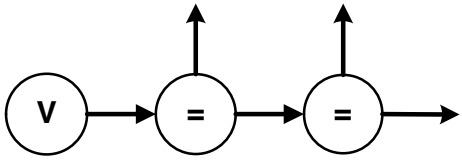

Fig. 3.6: Factor graph equivalent for junction of Type 0.

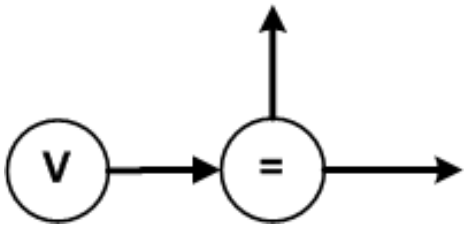

Fig. 3.7: Factor graph equivalent for junction of Type 0 with one of the output branches containing a signal source.

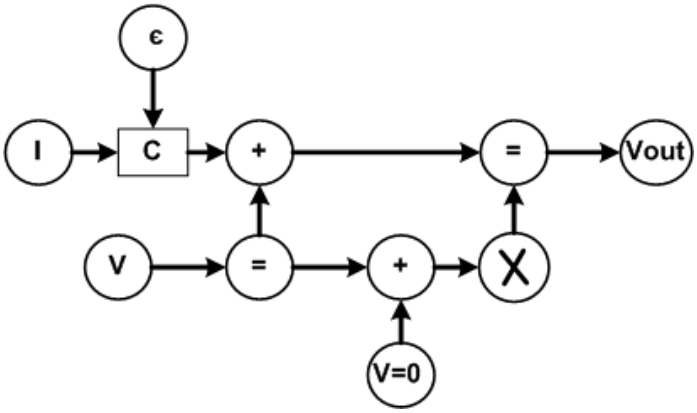

Fig. 3.8: Factor graph equivalent for the example circuit shown in fig. 3.5.

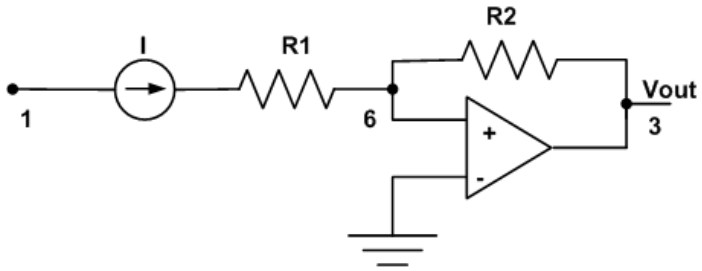

Fig. 3.9: Example circuit showing junction of Type 1.

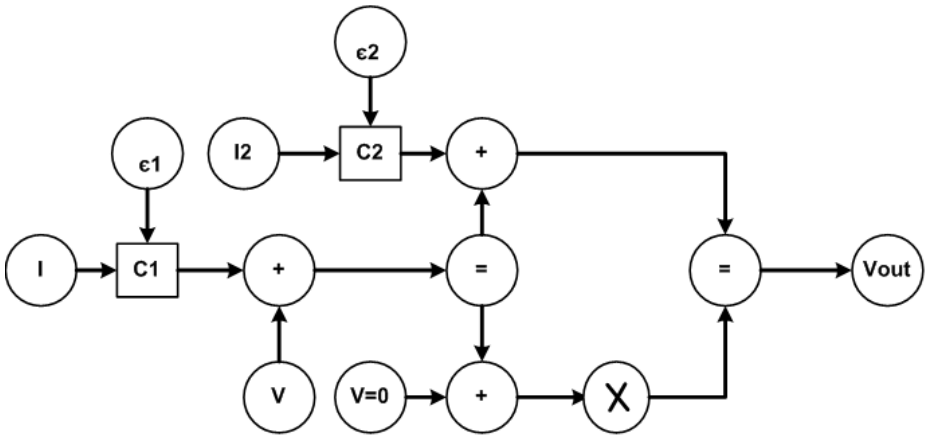

Fig. 3.10: Factor graph equivalent for the example circuit shown in fig. 3.9.

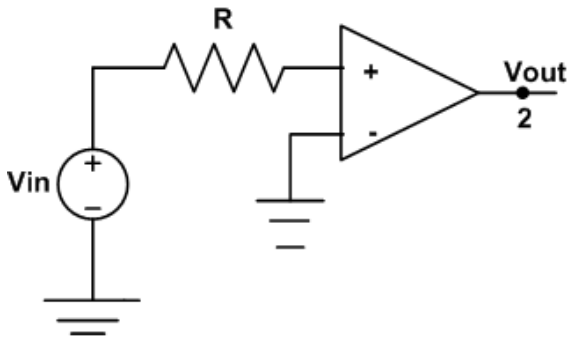

Fig. 3.11: Example circuit showing junction of Type 2.

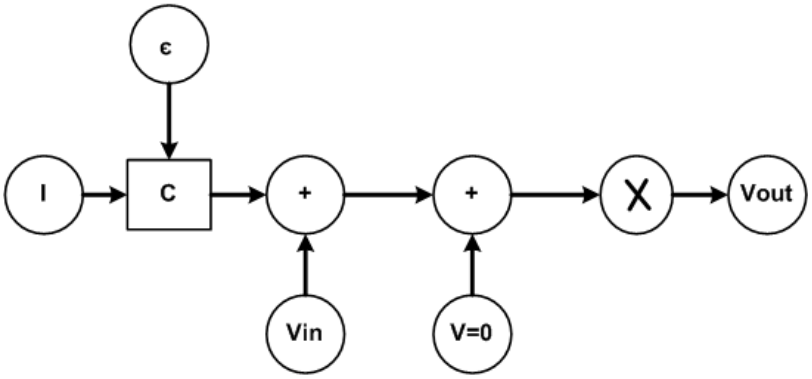

Fig. 3.12: Factor graph equivalent for the example circuit shown in fig. 3.11.

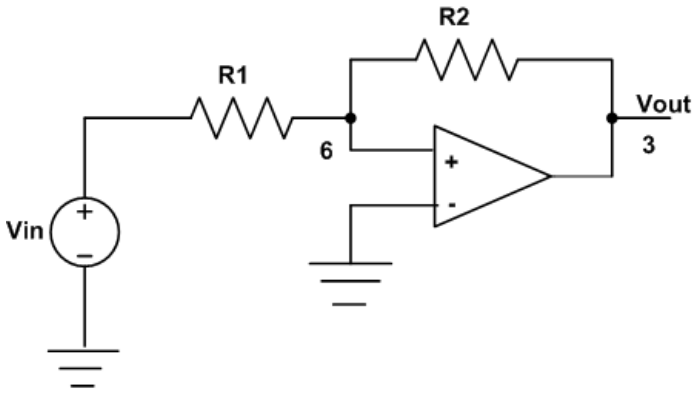

Fig. 3.13: Example circuit showing junction of Type 3.

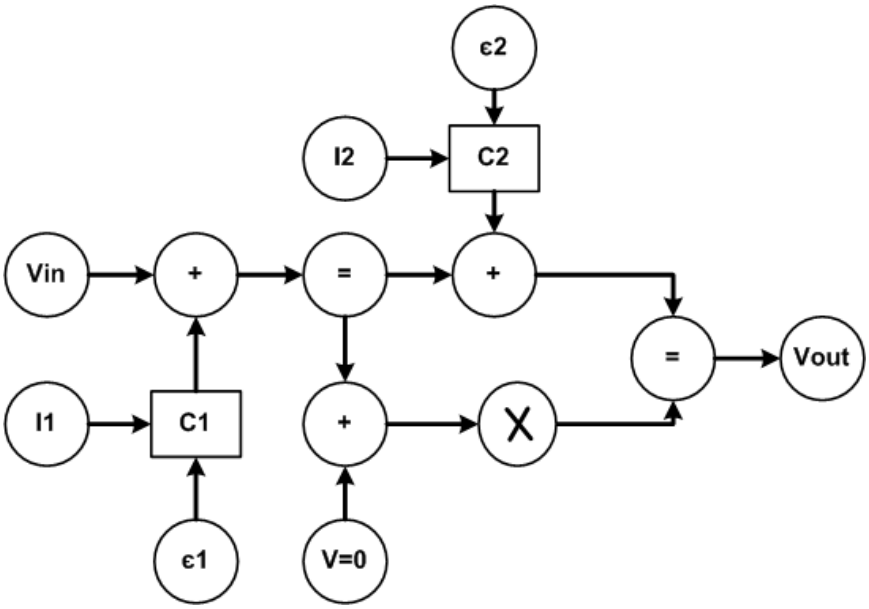

Fig. 3.14: Factor graph equivalent for the example circuit shown in fig. 3.13.

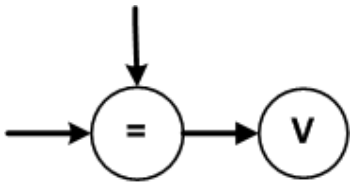

Fig. 3.15: Factor graph equivalent for junction of Type 3.

graph node to represent the junction voltage. Also, in case this junction represents the output node of a circuit, it is modeled by an output factor graph node representing that junction's voltage. An example circuit showing junction of Type 4 and its equivalent factor graph are shown in figs. 3.16 and 3.17, respectively.

- Type 5: This type corresponds to a junction with two inputs and one output. An example circuit showing junction of this type and its equivalent factor graph are as shown in figs. 3.18 and 3.19, respectively. Its equivalent factor graph block combination is shown in fig. 3.20.

- Type 6: It possesses one input and two output edges. An example circuit showing junctions of this type and its equivalent factor graph model are shown in figs. 3.21 and 3.22, respectively. Figure 3.23 depicts its corresponding factor graph model.

- Type 7: It is a junction with zero inputs and two outputs. An example circuit showing junction of this type and its equivalent factor graph are shown in figs. 3.24 and 3.25, respectively. It transforms into the factor graph block combination shown in fig. 3.26.

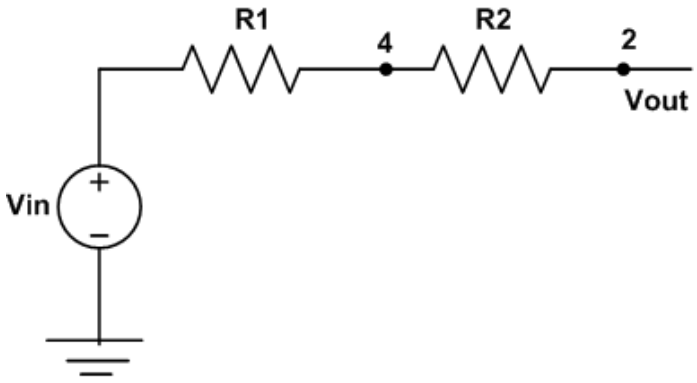

Fig. 3.16: Example circuit showing junction of Type 4.

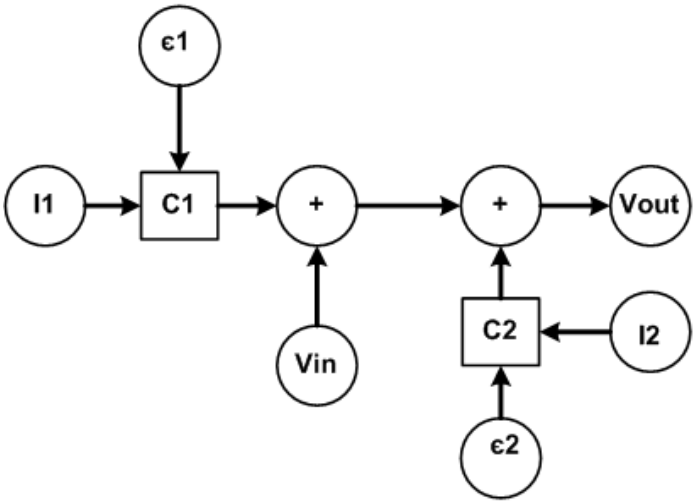

Fig. 3.17: Factor graph equivalent for the example circuit shown in fig. 3.16.

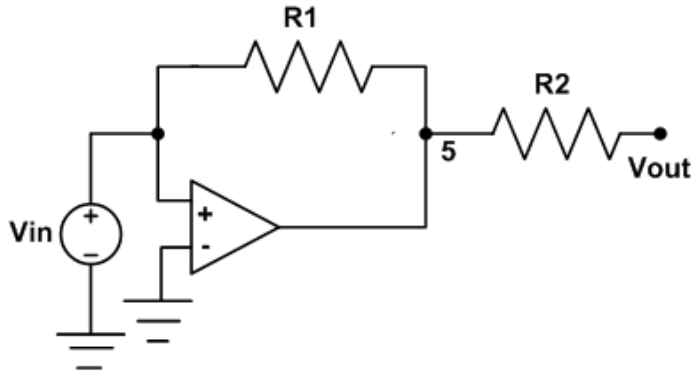

Fig. 3.18: Example circuit showing junction of Type 5.

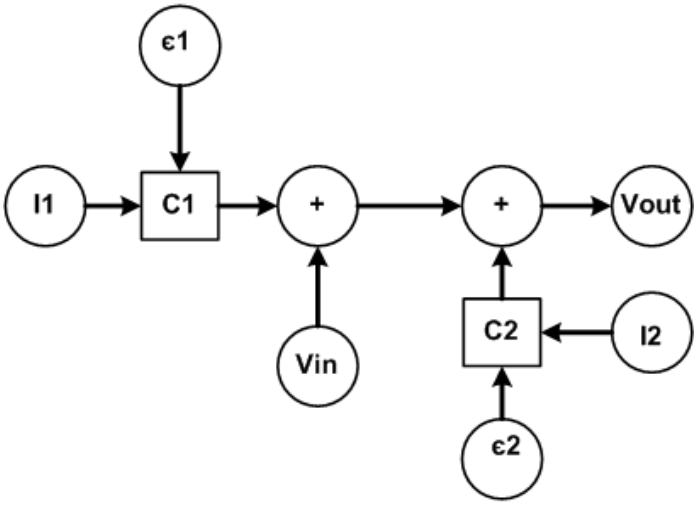

Fig. 3.19: Factor graph equivalent for the example circuit shown in fig. 3.18.

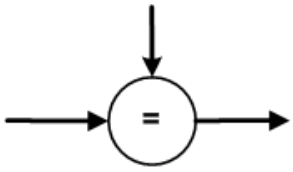

Fig. 3.20: Factor graph equivalent for junction of Type 5.

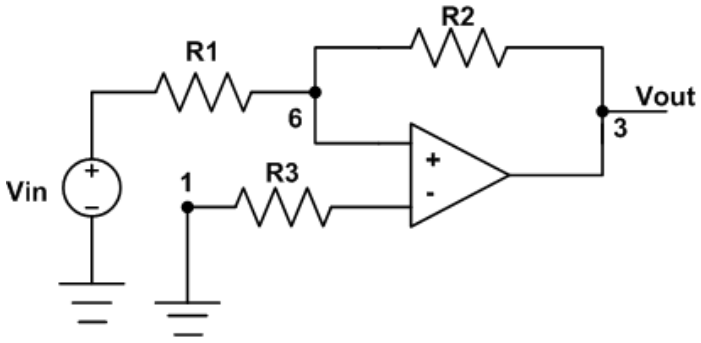

Fig. 3.21: Example circuit showing junction of Type 6.

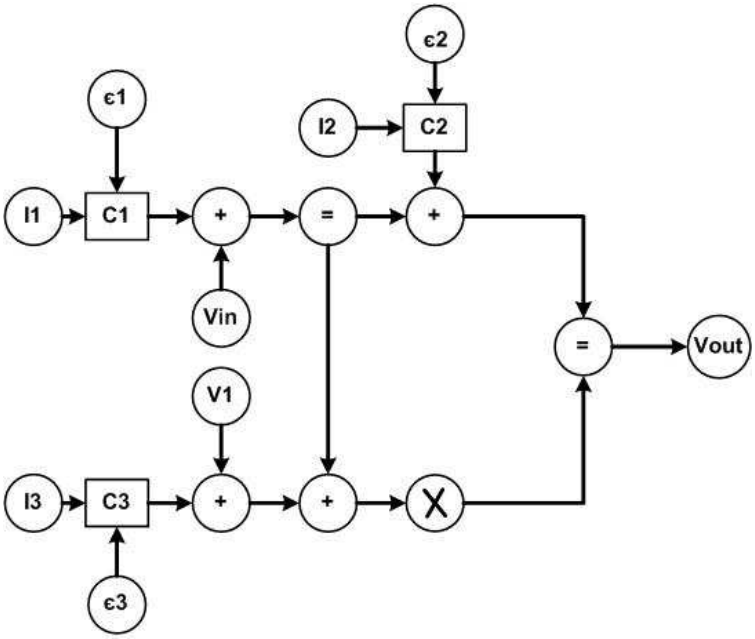

Fig. 3.22: Factor graph equivalent for the example circuit shown in fig. 3.21.

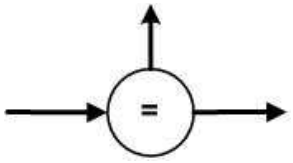

Fig. 3.23: Factor graph equivalent for junction of Type 6.

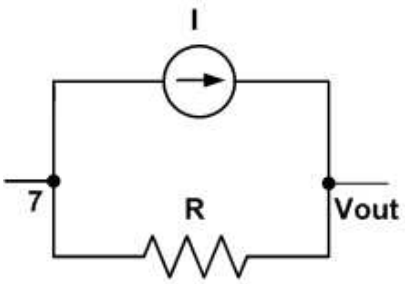

Fig. 3.24: Example circuit showing junction of Type 7.

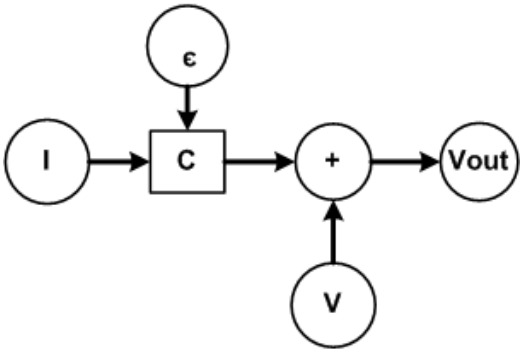

Fig. 3.25: Factor graph equivalent for the example circuit shown in fig. 3.24.

Incase, if one of the output branches contain a signal source, its equivalent factor graph model consists of an input factor graph node representing that junction's voltage.

- Type 8: This type of junction has two input edges and two output edges. An example circuit showing junction of this type and its equivalent factor graph are shown in figs. 3.27 and 3.28, respectively. Its equivalent factor graph model has been depicted in fig. 3.29.

All the rules discussed above convert the model of a physical circuit into a mathematical signal flow graph that incorporates all the basic electrical laws obeyed by the circuit in its structure. The factor graph technique shifts the circuit analysis domain from physical to mathematical while appropriately preserving the circuit's behavior. An example of a linear resistive circuit and its equivalent factor graph model are shown in figs. 3.30 and 3.31, respectively.

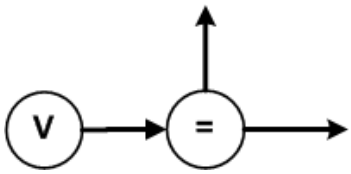

Fig. 3.26: Factor graph equivalent for junction of Type 7.

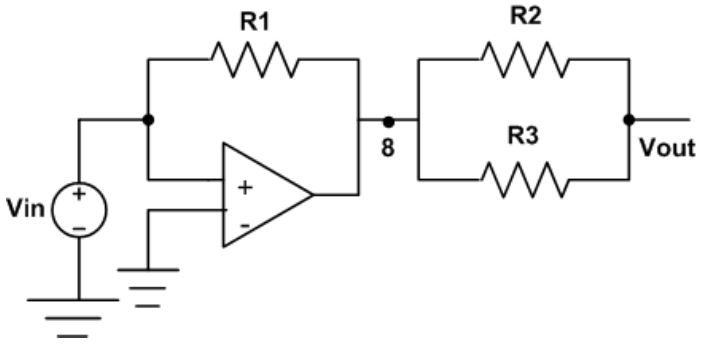

Fig. 3.27: Example circuit showing junction of Type 8.

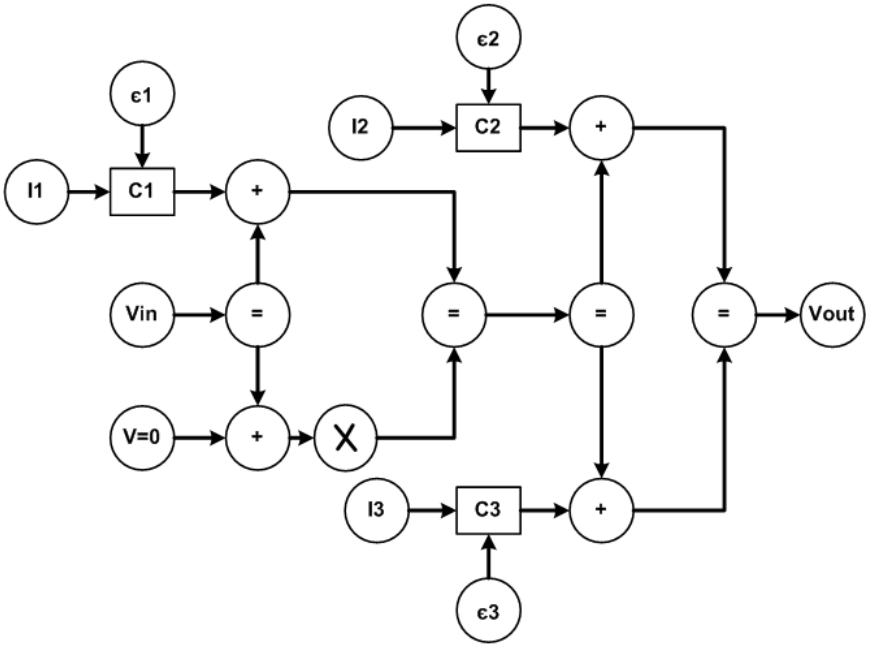

Fig. 3.28: Factor graph equivalent for the example circuit shown in fig. 3.27.

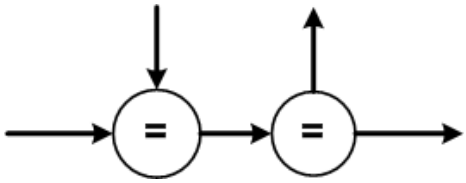

Fig. 3.29: Factor graph equivalent for junction of Type 8.

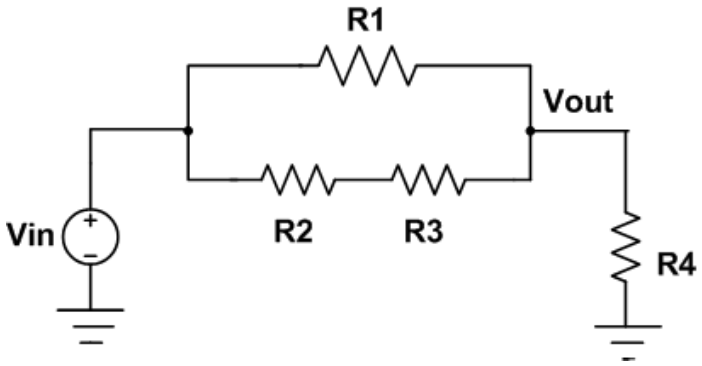

Fig. 3.30: Example of a resistive circuit.

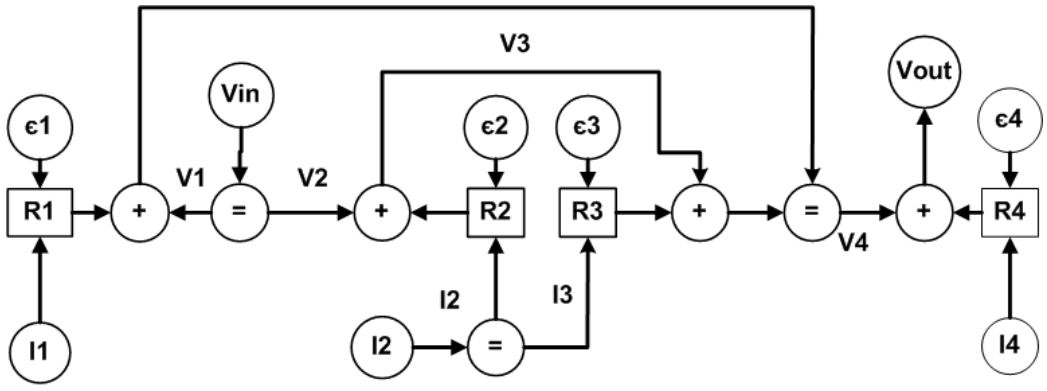

Fig. 3.31: Equivalent factor graph of resistive network shown in fig. 3.30.

3.2 Tool Flow

In this project, we have developed a tool prototype for implementing factor graph-based analysis of linear analog circuits based on the theory discussed in sec. 3.1. Given an analog circuit model, the tool chain facilitates the conversion of a circuit into an equivalent factor graph model and subsequent simulation to arrive at yield and parameter estimates. Figure 3.32 captures the basic structure of the tool.

The different steps involved in the tool chain are as follows.

- Firstly, the user creates analog circuit models in a design environment developed using a domain-specific modeling tool, Generic Modeling Environment (GME) discussed in Chapter 1. The metamodel configuring GME for drawing analog circuit models in the tool is designed using its Meta GME paradigm. It enforces the syntax dictating the circuit nodes and connections in the modeling environment.

- The next step in the chain involves running Monte Carlo simulations on the circuit model being analyzed. It is targeted towards generating thousands of circuit instantiations taking into account all statistical parameter variations and then running PSPICE simulations on all of them to evaluate the statistics of the circuit's output.

- The circuit model is simultaneously fed to another interpreter that interprets the circuit topology to generate a PSPICE input file and then runs SPICE simulator on it to evaluate initial current and voltage statistics along all the branches and the nodes in the circuit.

- The values obtained from the SPICE simulator are utilized by the next interpreter in the chain, which is responsible for translating the analog circuit model into its equivalent factor graph model. The translator interpreter is developed in C++ and contains all the previously discussed translation rules for different circuit elements coded in it. It accepts an analog circuit model as an input and generates corresponding factor graph output.

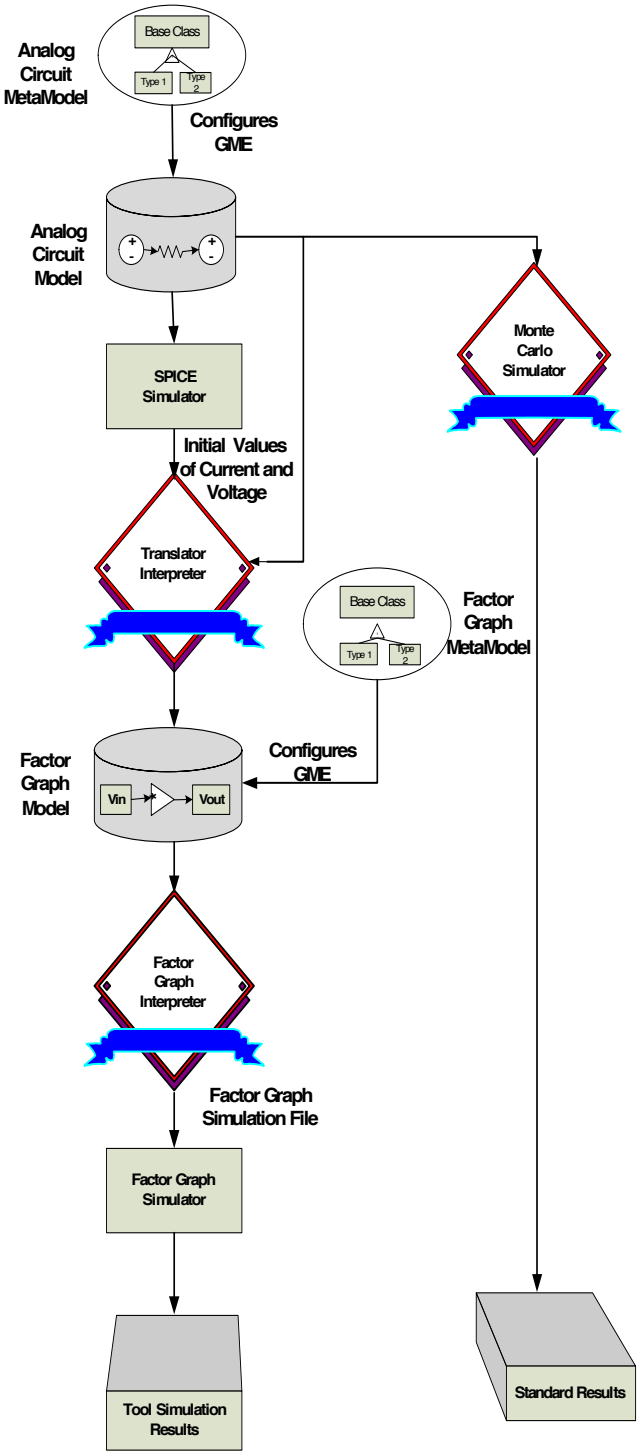

Fig. 3.32: Tool flow for the project.

- The factor graph model obtained from the translator interpreter is then fed as an input to the factor graph interpreter, which interprets the input factor graph model in order to realize the type of circuit nodes and their topology. It generates a log file at the output that contains all the factor graph information. The language used to develop this interpreter is C++.

- The final step in the tool flow involves executing the log file containing the factor graph information. A C++ based program for realizing the associated mathematics behind the factor graph is developed as a part of the tool chain. This program executes the various equations implementing the basic factor graph nodes, and passes messages as appropriate for the analysis. The result is a log file again providing the computed means and variances for each factor graph node.

The simulation results obtained at the end of the tool chain are compared against the standard results obtained from the Monte Carlo simulator to evaluate the accuracy of the tool.

Having been acquainted with the factor graph theory and tool structure so far, we proceed with a discussion of the design strategies for the analog circuit and factor graphs and the algorithms employed by the interpreters. Section 3.3 discusses these details and explains the way the tool structure has been implemented.

3.3 Tool Structure in Detail

Each of the five steps involved in the tool chain are discussed in more detail in this section.

3.3.1 Design Environment to Model Analog Circuits

The first step in the tool chain involves drawing an analog circuit model and embedding input signal values and parameters in the model such that the circuit specifications are completely captured. The design environment to model analog circuits is developed using

GME by utilizing its meta modeling capability. Figure 3.33 shows the top-level model of the paradigm for analog circuits.

The metamodel for analog circuits contains a model, named Container, which represents the container for an analog circuit design. Two model entities inherit from Container: Hierarchy and Analog_Circuit. Analog_Circuit contains all the circuit elements and their interconnections. The Hierarchy model is used to facilitate the partitioning of a design into subgraphs, where each subgraph is represented on its own sheet in the diagram. It facilitates better visualization of a large circuit and avoids clutter by dividing its different subcircuits into partitions. Each subcircuit could be further partitioned into even smaller sub subcircuits as per the size of the model. The containment relationship of the hierarchy entity within the container enforces the rule that a partition could be contained in both the inherited models of the container: Hierarchy and Analog_Circuit. Having set the rules for laying out a circuit in the model, the containment relationship of analog-element within the container provides the ability to contain an analog element in a partition (or subcircuit) or an analog circuit in the model. Attributes define the properties of the objects. Users specify their values at modeling time. The four attributes of an analog element include MeansOfMixture, DensitiesOfMixture, WeightsOfMixture, and VariancesOfMixture. The values of these attributes are specified by the user to represent the Gaussian signal associated with that element. The connections between different circuit elements are modeled by the connection objects contained in the container that simulate wires in the physical circuit. Connectors are the ports residing in an analog element that facilitate the connection between two elements through them. Sign_Neg is an attribute of the connector that can be set by the user to either a true or false value so as to specify the polarity of the signal value associated with that connection. Sign_Neg is particularly relevant to an analog element type such as an amplifier to differentiate between its negative and positive input terminals.

To facilitate connections between two analog elements contained in different partitions, reference to an element can be created into another element's partition and connection

between these two can be established. A reference is a copy of the element without any reallocation of memory space to it. In the metamodel, Element_Ref is a reference to the analog element that could be contained within the container. Different types of analog circuit elements inherit from the base class of Analog_Element. These include all kinds of analog and digital circuit components required to build analog and mixed signal circuits. Owing to the inheritance relationship, the types share properties of the analog-element in common and also posses their own additional attributes. The abstract attribute of Analog_Element class is set to true in the metamodel making it invisible in the modeling environment. Only the inherited types are available to the user in the design catalog to build circuits. Figures 3.34, 3.35, 3.36, and 3.37 depict the various inherited types of circuit elements in the metamodel.

ANALOG COMPONENTS

1. Independent Sources: Active devices such as independent current and voltage sources can be modeled using objects called IndepCurrentSource and IndepVoltageSource. They possess an attribute named SignalSource which could be specified as true by the user depending on whether a source acts as a starting input to the circuit.

2. Amplifier: Amplifier is used to model a two input terminal amplifier with a gain attribute to specify the open loop gain of the amplifier.

Figure 3.34 shows these two kinds of analog circuit components in the metamodel.

3. Junction: Junction object represents a node or a junction in an analog circuit. Its CommonGround attribute indicates whether the junction represents ground voltage and the IsOutputnode attribute indicates whether the output of the circuit is desired to be obtained at that junction. Both these values are specified by the user in the model according to the circuit requirement. Its other attributes such as RequiredOutputMeans, RequiredOutputVariances, RequiredOutputWeights, and RequiredOutputDensities correspond to the Gaussian signal that represents the desired yield required to initiate backward propagation from that junction, in case it is chosen to be an output node. These values are supplied by the user in order to calculate a parameter estimate starting with the required yield as input.

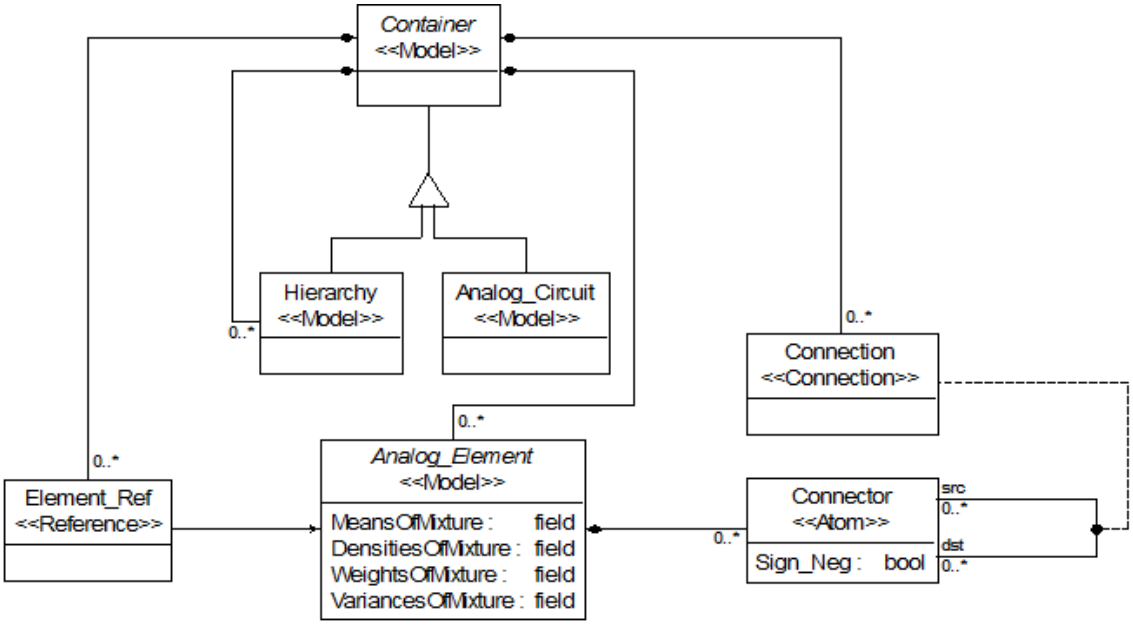

Fig. 3.33: Metamodel for an analog circuit in GME showing the root of the structure and its inherited components.

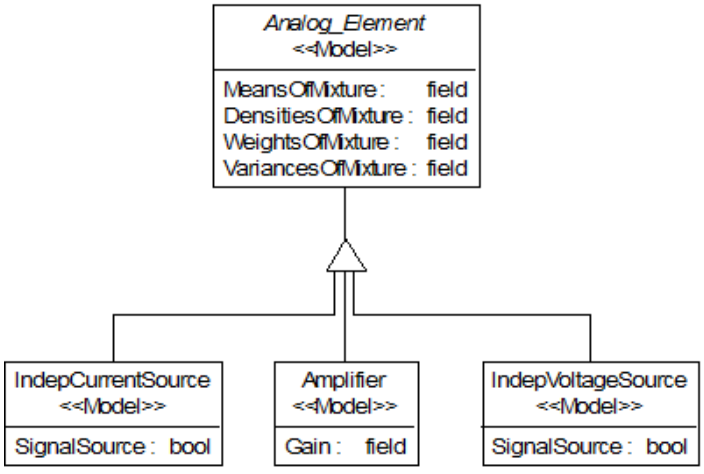

Fig. 3.34: Metamodel for analog circuits depicting different kinds of analog circuit components such as amplifier and independent signal sources.

NodeId and VoltageValue correspond to the attributes that are automatically set by the interpreter during the translation process. NodeId represents a unique ID associated with each junction in the network and is utilized by the translator interpreter later while the voltage value specifies the voltage associated with that junction. This value is read by the translator interpreter from the SPICE output file obtained by running SPICE simulation on the circuit model. Figure 3.35 shows the junction metamodel.

4. Resistor: The Resistor class is used to model a resistance in the circuit. A resistor maps onto the combination of a coefficient block and an error source node in the factor graph domain. The attribute named Resistance_Value represents the value of the resistor. The user could specify the error tolerance in the value of a resistor as a Gaussian distribution through RFMean, RFVariance, and RFWeight attributes of the resistor. Its other attributes, that include NodeId, CurrentSourceMean, CurrentSourceVariance, and CurrentSourceWeight are automatically set to appropriate values in the translation process and need not be specified by the user. The values of CurrentSourceMean, CurrentSourceVariance, and CurrentSourceWeight, corresponding to the Gaussian current signal propagating in the branch containing that resistor, are obtained from the SPICE output file. These values are then communicated to the resulting factor graph by the translator interpreter. Figure 3.36 shows the resistor metamodel.

5. Dependent Sources: The classes named DependentCurrSource and DependentVoltSource in the metamodel represent the four types of dependent sources discussed in sec. 3.1. Both these objects have an attribute termed TypeofDependency, which can be set to voltage or current based on the requirement to model voltage-dependent or current-dependent sources. The Transconductance, Transresistance, and Factor attributes of these sources allow the user to specify the dependency factor based on the type of dependency. The dependent sources contain references to the objects: Junction and Resistor, named JunctionRef and ResRef. This allows the user to specify the two junctions whose difference of voltages determines the output signal values obtained from the voltage dependent sources by including the references to the junctions within the sources. It also enables specification

of the current signal which determines the output values of current dependent sources by using reference to a resistor object that carries the desired current through it. These are depicted in figs. 3.35 and 3.36.

DIGITAL COMPONENTS

These include Comparators, Multiplexors, and Switches. There has not been much exploration in the analysis of digital and mixed signal circuits with this tool and so this area possesses a lot of potential for research and extensibility. Figure 3.37 shows different kinds of digital circuit components included in the analog circuit metamodel.

The metamodel discussed above can be further extended to include more circuit components and add more attributes to the existing ones to be able to widen the scope of the tool for modeling varied kinds of circuits. Two examples illustrating the circuit models designed using the tool are shown in figs. 3.38 and 3.39.

3.3.2 Modeling Paradigm to Design Factor Graphs

The factor graph equivalent of an input circuit model is generated as a result of the translation process in the tool chain and is abstracted away from the user. This process conforms to the rules defined in its metamodel. The metamodel for a factor graph is shown in parts in figs. 3.40, 3.41, and 3.42. It defines all the components of the factor graph domain in a hierarchical structure with the Parent object at the root of the hierarchy. This is analogous to the Container object in an analog circuit that serves as the base for designing graphs. The objects named Graph and Partition are inherited from the Parent to define different levels of hierarchy in the factor graph. A Graph object represents the layout containing all the graph nodes and the connections on it, similar to an Analog_Circuit object. The Partition corresponds to a Hierarchy object in the analog circuit metamodel that allows one to decompose the graph into different layers to avoid visual clutter. The Node object contained within a Parent is used to model different types of factor graph nodes. Its attributes include Means_Messages, Variances_Messages, Weights_Messages, and Weights_Densities that represent the Gaussian signal associated with that node. These

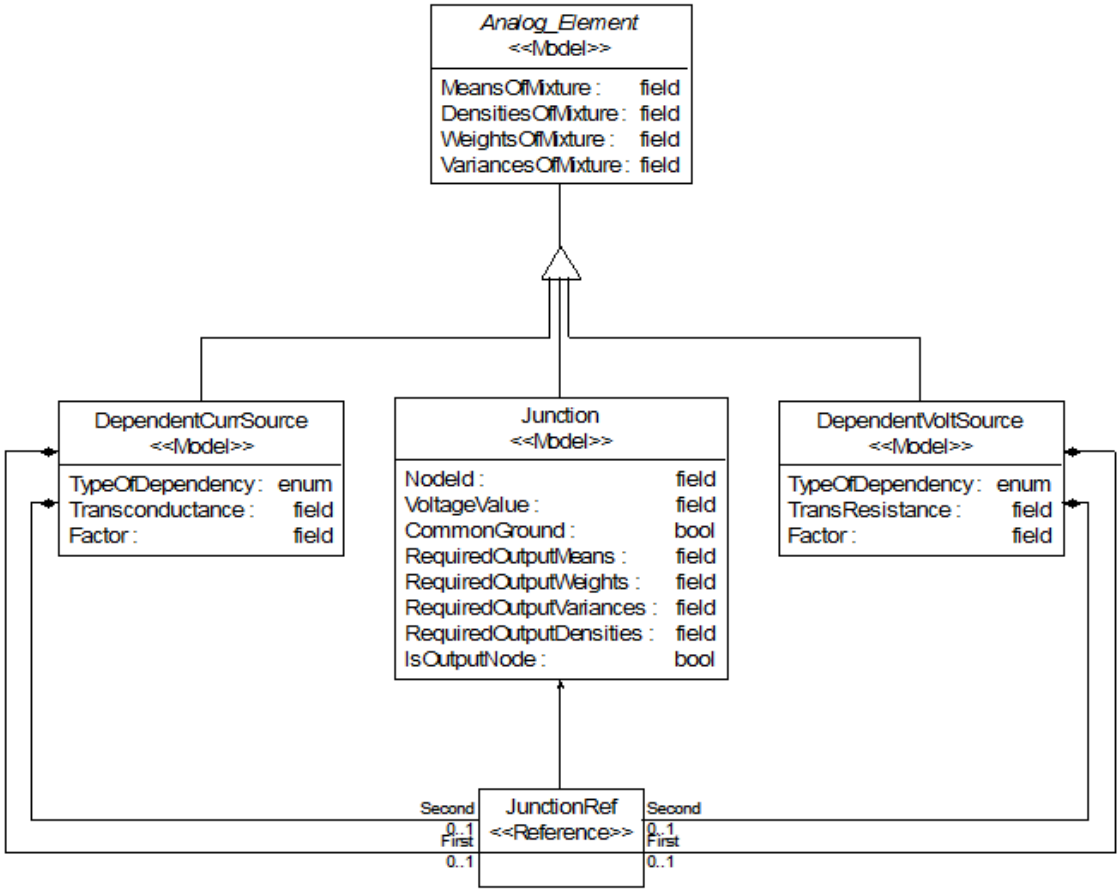

Fig. 3.35: Metamodel for analog circuits showing dependent sources and junction.

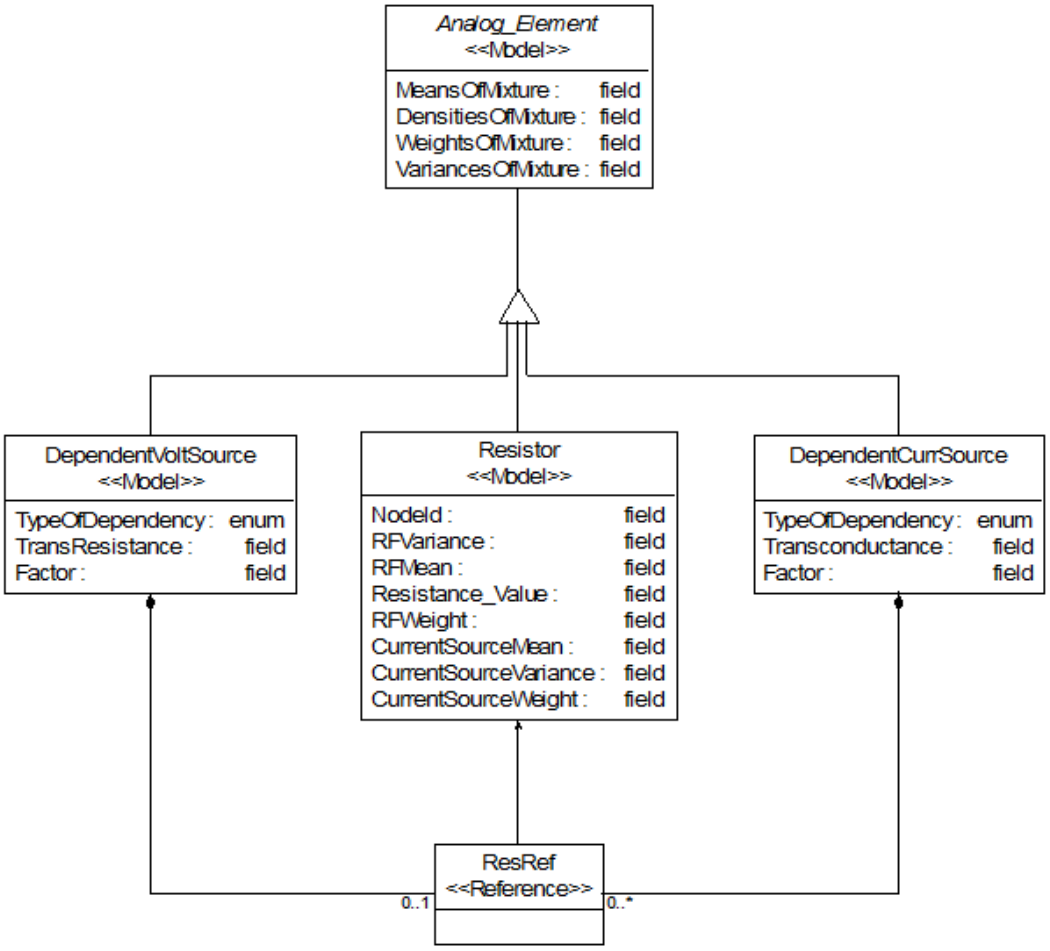

Fig. 3.36: Metamodel for analog circuits depicting dependent sources and resistor.

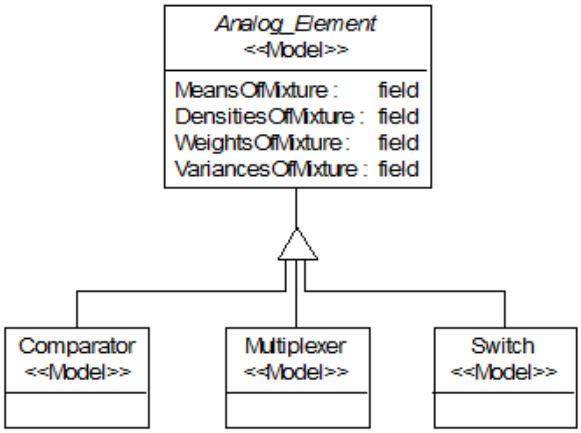

Fig. 3.37: Metamodel for analog circuits showing digital components available in the design environment.

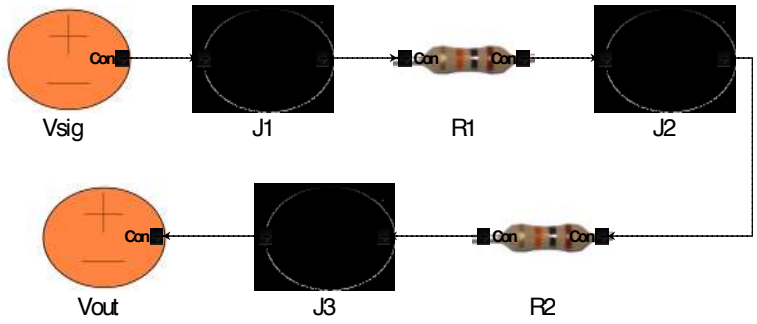

Fig. 3.38: Example of a linear analog circuit model drawn in GME.

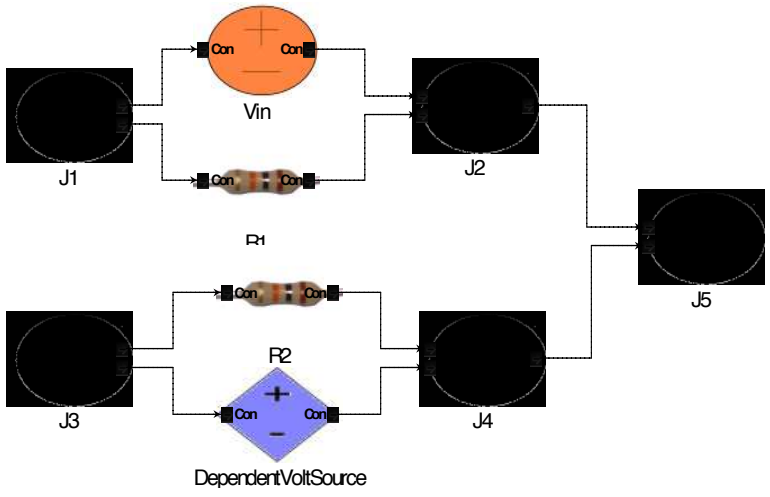

Fig. 3.39: Analog circuit model drawn in GME containing a dependent voltage source.

values are populated in the factor graph model by the translator interpreter from the corresponding attributes of the analog circuit element in the circuit model. Seven different kinds of factor graph nodes inherit from the Node object, each of which bears relationship with an element in the analog circuit domain. The abstract attribute of the Node object is set to true to make it unavailable and have only the inherited types appear in the factor graph design environment. The Noderef is a reference object referring to a Node in the graph that could be contained in a Parent. It enables the connections between two nodes existing at two different levels of hierarchy. The connections between different node objects are provided by the Edges contained in the Parent. The Edges connect to the Ports contained within the Nodes. The Ports map onto the Connector objects in a circuit while Edges correspond to the Connections in an analog circuit. Negate is an attribute of ports that defines the polarity of the signal associated with that Connection and depends upon the user specified value for the Sign_Neg attribute of the Connectors in the circuit model.

The different types of factor graph blocks are described below.

1. Input_Output: The Input_Output block represents either an input or an output entity of a circuit and typically models all the voltage and current (signal) sources in the circuit model. It acts as the starting point for the Gaussian message propagation along the edges of the graph. Its Type attribute indicates the type of the source to be either voltage or current.

2. Multiplier: This node is a one input block that defines equations to perform multiplication of the Gaussian signal fed to it as input. It possesses two attributes termed MultiplierType and Coefficient. The Coefficient represents the multiplication factor while MultiplierType is an enumerated list with Scalar, Vector, and Matrix as its three members defining the three different types of the multiplier nodes. In case of a scalar multiplier, the Coefficient is a scalar value, for the vector multiplier it is a vector and the Coefficient is a matrix for the matrix multiplier. A scalar multiplier is used to model the amplifier component of the analog circuit with its Scalar Coefficient representing the gain factor of the amplifier.

3. Equality_Constraint: This node allows for the distribution of Gaussian signals almost equally along different branches in the graph. It could have either one or two inputs and is typically used to model junctions in an analog circuit.

Figure 3.41 shows these three kinds of factor graph nodes in the metamodel.

4. Coefficient and Error_Source: The combination of these two nodes represents a Resistor in the factor graph modeling the parametric variations in the resistance value. The Factor attribute of the Coefficient node corresponds to the resistance value specified by the user in the circuit model.

5. Adder: It is a two input node and defines equations for the addition or subtraction of its input Gaussian signals depending on the attributes of the ports specified by the user. It is typically used as a part of the combination that models Ohm's law obeyed by a resistor in the circuit and binds the voltage and current subgraphs together.

6. Bit_Probability_Estimator: This node forms a part of the graph corresponding to a mixed signal circuit model. It defines equations to obtain the probability of the input analog Gaussian signal of representing either bit 0 or bit 1 in the digital domain. It is responsible for converting an analog signal in a particular range to a binary bit.

Coefficient, Error_Source, Adder, and Bit_Probability_Estimator nodes are depicted in fig. 3.42.

There exist well-defined equations for each of these nodes that guide the forward and backward Gaussian message propagation through them. They are borrowed from Loeliger's work [10] except for the coefficient and error source blocks. The equations for these two blocks have been devised intuitively based on the mathematics involved in the propagation of a current signal through a resistor. All these equations are compiled in Appendix A.

3.3.3 SPICE Simulator

The SPICE Simulator is the first interpreter in the tool chain that employs a SPICE simulation on the user-specified circuit model to ascertain initial mean values for voltages and currents for each junction and resistor in the circuit. It interprets the input circuit model to resolve its elements and connections and generates a PSPICE input script file

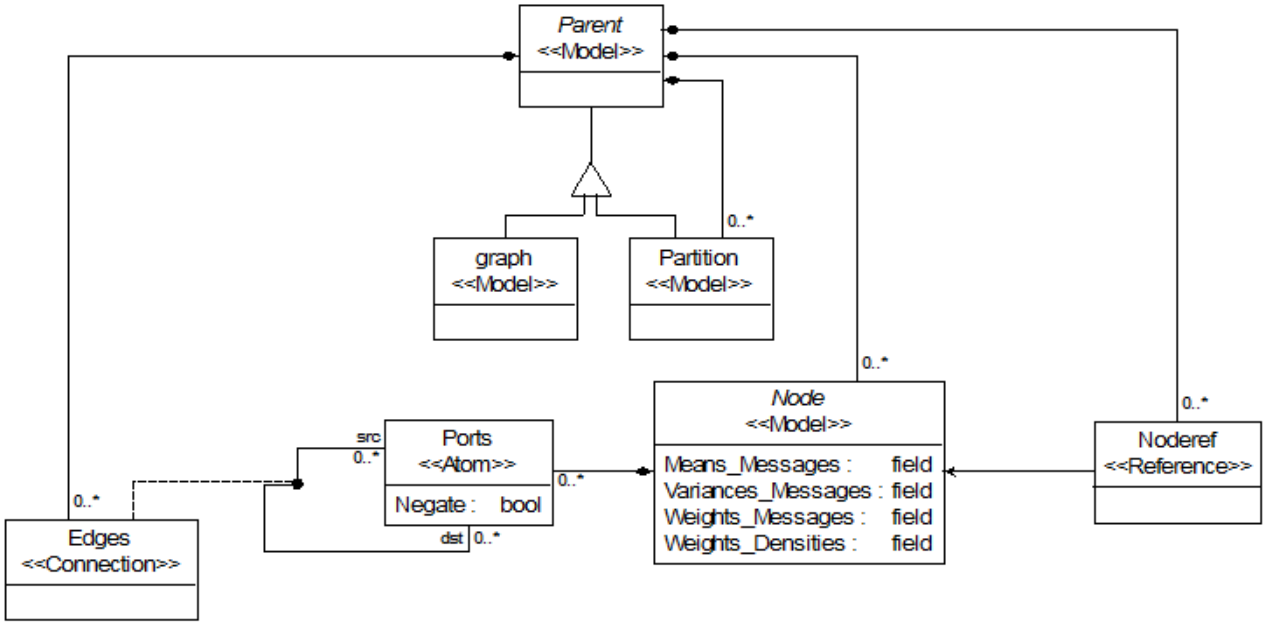

Fig. 3.40: Metamodel for a factor graph showing the root of the structure and its inherited elements.

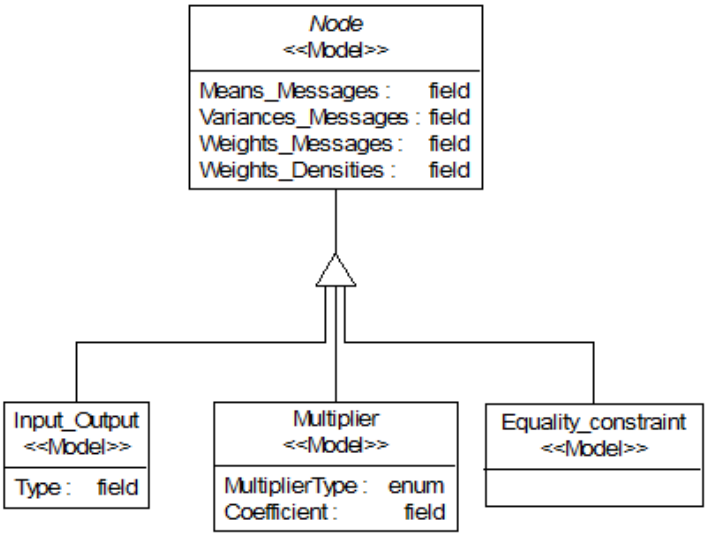

Fig. 3.41: Metamodel for a factor graph depicting different types of factor graphs.

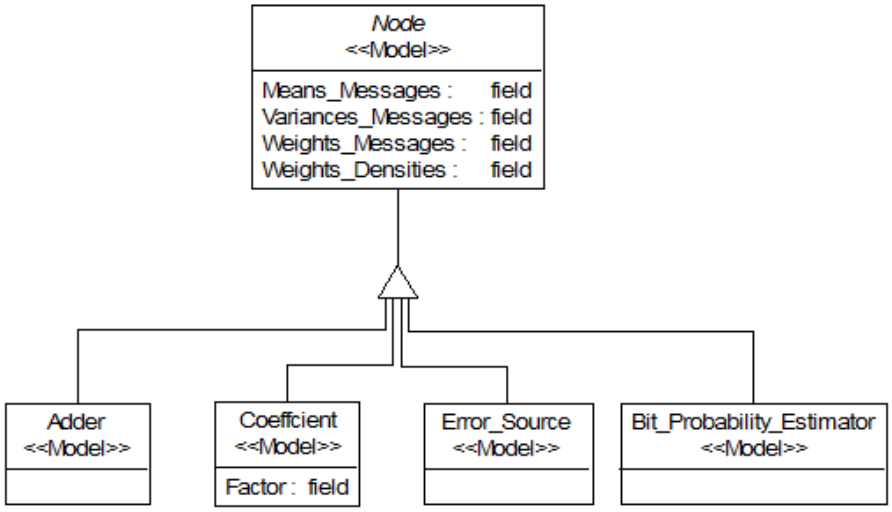

Fig. 3.42: Metamodel for a factor graph showing different types of factor graphs.

describing the circuit topology. The rules for generating a PSPICE script file in the correct executable format have been derived from few textbooks and websites on PSPICE [21–24]. The interpreter then executes PSPICE simulator with the generated SPICE file as an input to obtain a SPICE output file listing the values of current and voltage across each resistor and node in the circuit. This algorithm is repeated several times to generate multiple instantiations of the SPICE output file for the same circuit, taking into account parametric variations. The interpreter finally gathers statistics from all the output files to derive initial values for the mean and variance of the current and voltage signals at different points in the circuit. These values are then propagated to the voltage and current attributes of the resistor and junction objects in the model to be utilized by the translator interpreter later in the tool chain. The SPICE simulator is particularly useful to determine the initial values of current signals across each resistor in the circuit.

3.3.4 Translator Interpreter to Obtain a Factor Graph from an Analog Circuit Model

The next step in the tool chain involves transformation of the circuit model into its equivalent factor graph, conforming to the translation rules discussed in sec. 3.1. This task

is accomplished in software through a translator interpreter program. The program accepts an analog circuit model as an input and applies a translation algorithm to it to obtain a corresponding factor graph output. It reflects the basic class structure represented by the metamodel and utilizes the API's associated with GME objects to gather information about the model. The translator interpreter is developed in C++ to utilize its STL library of data structures to organize a collection of similar objects in the model into discrete sets.

The interpreter starts by resolving the hierarchical structure of the circuit model to collect the set of all the analog circuits in the root folder that exist at the root of the hierarchy. It then creates an empty factor graph for each of the circuits in the factor graph model. It also gathers all the levels of hierarchy in each of the circuits together and creates partition objects in the corresponding factor graph. In order to keep a record of the containers in both, the analog circuit and the factor graph domain, the interpreter maintains a map to insert a pair of analog circuit container objects and their analogous factor graph parent objects. It then visits each container in the analog circuit individually and collects all types of analog circuit elements and edges (or connection objects) contained in it into two separate lists. The list of circuit elements is parsed to create equivalent factor graph blocks for each of them conforming to the rules of translation discussed before. The factor graph objects are created in the parent object of the graph which is obtained from the map, using a container object as a key that contains the analog circuit element analogous to that factor graph object. The interpreter maintains another map to insert a pair of analog elements and a list containing their corresponding factor graph block combinations in it. For every analog circuit element, an entry is made in the map. There are functions defined to specify the name, mean, variance, weight, and density attributes of the Gaussian signal associated with the generated factor graph node. This information is gathered from the attributes of analog circuit element objects and propagated to the corresponding attributes of its factor graph nodes. It also collects all the references in the circuit at each level of hierarchy and derives the analog element being referred to by that reference in the circuit. It then obtains the corresponding factor graph block for that element from the map and creates its reference

in the factor graph model. The interpreter processes all circuit elements in the model except junctions at this point and keeps a record of all the junctions in a separate list. Once all the circuit elements are processed, it revisits the list of junctions and creates nodes for each of them. It first sets the type for each junction by exploring its input and output edges and then creates an object of type "junctionnode structure" to encapsulate all the information about the junction's input and output elements within it. The data structure also possesses a few flags as its members to indicate if a junction is common ground or desired to be an output node. There is a map that contains a pair of junction objects and their junctionnode data structures for every junction within it. This is useful later when creating connections to obtain the corresponding junctionnode structure from the map for a particular junction object.

After creating factor graph blocks for all circuit components in their respective containers, the interpreter sets out to establish connections between these blocks according to the circuit topology. In order to perform this task, it parses the connections in the circuit and for each connection object, input, and output elements connected to it are determined. For these elements, it derives the equivalent factor graph blocks from the map and connects the two in the resulting factor graph by creating an object of class edges in the graph. The objects named "Ports" are analogous to the connectors in the analog circuit and created within factor graph blocks to facilitate the connections between them.

3.3.5 Factor Graph Interpreter to Organize the Information of the Graph in a Log File

When the translator interpreter generates the factor graph, the factor graph interpreter is run as the next step in the tool chain. It is responsible for resolving the hierarchical structure of the factor graph model and dumping it into a log file. It adopts a top-down approach to decompose the graph into its individual elements, similar to the translator interpreter. To begin with, the factor graph interpreter obtains the set of factor graph objects contained in the root folder and then proceeds to gather all the constituent components of each of the graphs. It organizes all the factor graph nodes and partition objects in the graph into two

separate sets. It then parses the list of partition objects and adds the factor graph nodes in each partition to the existing set. After organizing all the nodes of the graph present at each level of hierarchy, it creates an instance of class structure, named "element," for each of them to bind together all the information related to a node . The members of this class structure include: name of the node, its type among seven inherited types discussed in the factor graph metamodel and variables storing the values of mean, variance, weight, and density attributes of the factor graph node. The element class also includes a member function that resolves the input and output edges of a node into two lists that form a part of its data members. An element object is created for every factor graph node in the graph. The interpreter then collects all the edges in the graph and creates edge objects for each of them. After decomposing the entire graph into its individual components, the interpreter resolves the topology of the graph to determine the order in which the graph components are connected so as to arrive at the output node starting with the input. It keeps track of this generated schedule by storing all the nodes in an ordered list. The last task of the interpreter involves generating a log file and dumping all the information gathered from the graph into it. The result of the factor graph interpreter is a C++ based program to realize the associated mathematics behind the factor graph. This file is executed in the last step of the tool chain to perform all the forward and backward message calculations for each component of the graph. The structure of the C++ log file includes array declarations of type float assigned to the values of mean, variance, weight, and density attributes of each node in the graph. The array declarations allow specifying multiple Gaussian signals to form a Gaussian mixture associated with one node. It also comprises of object definitions of different class types representing each node in the graph and appropriately assigns the values of node attributes to the corresponding data members of the objects. Finally, functions evaluating the statistics of the signals propagating through the graph nodes are called for each node in the order determined by the schedule. An example of this file for a circuit of an inverting amplifier is shown in Appendix B.

3.3.6 Final Simulation to Obtain Yield and Parameter Estimates

An entirely separate C++ project is responsible for executing the program to simulate the factor graph and arrive at the desired results. All the mathematics required to evaluate signals propagating through the graph is embedded within functions of the classes defined for every node in the graph. Each class corresponds to an inherited node type in the factor graph metamodel and inherits from a base class named component. They share in common the properties of the component class such as floats indicating mean, variance, weight, and density attributes of the nodes. The other data members include eight message pointer vectors pointing to the sets of incoming and outgoing edges from a node. The vectors define message propagation in both directions, forward, and backward. There are separate vectors to represent messages at the current and next time steps of the simulation. A message constitutes a class type to encapsulate floats representing mean, variance, weight, and density attributes of the nodes to be able to specify a Gaussian signal completely. The classes define member functions to evaluate the mathematical equations associated with each factor graph node discussed by Loeliger in his paper [10]. These functions allow message calculation for forward and backward propagation in the graph. The factor graph simulation algorithm is an iterative scheme employed to arrive at statistics associated with each node in the graph. During every iteration of the process, forward and backward message propagation functions are called for every node and new values are assigned to the next time steps' message vectors based upon the values of the vectors at current time step. Once all the nodes are processed, an "UpdateTimeStep" function is called that swaps the values of the message vectors at current and next time steps and evaluates the difference between these two sets of values. The process is terminated if the difference lies within a specified tolerance limit otherwise iterations ensue. The result of the simulator program is a log file providing the computed means and variances for each factor graph node, which consists of the results from the tool for yield and parameter estimates.

3.4 Monte Carlo Simulator

There is another interpreter involved in the technique which is completely separate

from the tool chain and produces Monte Carlo simulations of the circuit being analyzed. It accepts an analog circuit model as an input and generates thousands of circuit instantiations, taking into account all statistical parameter variations. It then runs PSPICE simulations for all of them to evaluate statistics of the circuit's output. These results are used as reference to evaluate the accuracy of the tool.

Chapter 4

Simulation Results Obtained by the Tool

This chapter presents the results obtained from the application of the tool to a few analog circuit models, showing its efficiency in performing the desired analysis. In order to validate the factor graph-based simulation results for the chosen circuits, corresponding results obtained from the Monte Carlo technique are taken as a reference. The accuracy and sensitivity of the analysis is examined with respect to changes in parameter values and input parameter statistics. The analysis results are plotted in MATLAB to quantify the comparison. These plots indicate that the tool's results approximately match with Monte Carlo simulations. Time efficiency of this method is compared by observing the execution time. The tool evaluates all the circuits that could result from the fabrication process simultaneously, while the Monte Carlo technique collects statistics for each circuit instantiation separately in thousands of simulation runs. The parallel processing of multiple circuits by the tool reduces the computation time by multiple orders of magnitude. The time measurements depicted in sec. 4.2 clearly indicate that an execution run of the entire tool chain requires few seconds as opposed to several minutes taken by Monte Carlo technique to arrive at yield and parameter estimates of a circuit. The tool thus appreciably surpasses Monte Carlo simulation on the grounds of total CPU time utilized.

4.1 Accuracy Results

Multiple simulation/analysis runs have been conducted with the tool to compare the results against Monte Carlo simulations for four different kinds of analog circuits. They include inverting and non-inverting configurations of an amplifier, an instrumentation amplifier and the small signal equivalent model of a MOSFET. All of these circuits were modeled with the available blocks in the design environment of analog circuits. These cir-

cuits follow from the definitions given in the textbook of microelectronics [25]. This section thoroughly discusses the plots obtained for all of the circuits mentioned above by applying the tool and Monte Carlo simulation.

4.1.1 Inverting and Non-Inverting Configurations of an Amplifier

The circuit diagrams of the two amplifier circuits are shown in figs. 4.1 and 4.2. In both of these circuits, the values of resistors R1 and R2 are varied in turn for all the statistical calculations performed by the tool. Two sets of analysis are conducted in the experiments: forward runs aimed at calculating output statistics for a specified set of inputs by varying parameters of resistances, and backward runs targeted at predicting unknown parameter values for a known yield constraint. The inherent variations in the resistance values owing to the fabrication process are accounted for by associating a Gaussian signal with each resistor and varying its variance over a specified range, centered around a zero mean. The parameter values of the resistors R1 and R2 are statistically varied over the range of 0.1 to 1e-10 for all the experiments performed. The values of the resistors chosen for the set of plots showing results obtained by varying the variance of resistor R1 are as shown in table 4.1. Table 4.2 lists the values of the two resistances that yield the results obtained by the parametric variations in the feedback resistor R2 over the specified range. The values of the resistors are chosen in such a way that in both the cases, the closed-loop gain of the inverting amplifier circuit is -10 and that of the non-inverting amplifier is 11. The open-loop gain of both the amplifiers for all calculations is set at 100K. The plots show the values of output statistics and parameter estimates as a function of the variance of resistors R1 and R2. The results obtained from the experiments have been a good measure of the accuracy of the tool in calculating the unknown yield and parameter estimates.

1. Experimental Results for Forward Propagation to Calculate the Yield: In order to

Table 4.1: Values of resistors chosen for the set of experiments performed by varying R1.

Resistor	Value
R1	1K
R2	10K

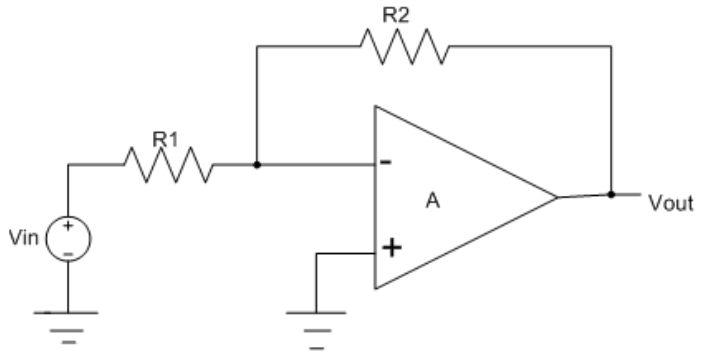

Fig. 4.1: Circuit diagram of an inverting amplifier.

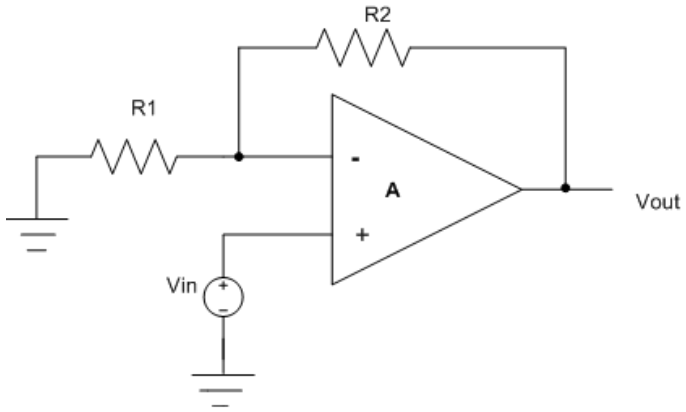

Fig. 4.2: Circuit diagram of a non-inverting amplifier.

Table 4.2: Values of resistors chosen for the set of experiments performed by varying R2.

Resistor	Value
R1	100ohm
R2	1K

set up a formal mode of comparison, a statistical analysis of the yield is conducted using both the factor graph technique and the Monte Carlo simulation, by varying the value of the variance applied to the resistors R1 and R2 in each of the two circuits. Each technique results in a specific computed mean and variance of the output voltage. The values of output statistics are plotted against the variation in the parameters of resistances R1 and R2 for both the techniques. Figures. 4.3 and 4.4 show the plots of mean and variance of the Gaussian output signal, Vout, against statistical variations in resistance R1 obtained by each technique for both the circuits. For this set of experiments, the variance of feedback resistor R2 is fixed at a value of 1e-4. The plots in figs. 4.3 and 4.4 indicate the accuracy of the tool with respect to the Monte Carlo simulation.

Figures 4.5 and 4.6 depict the mean and variance values of the output signal obtained by varying R2's variance over the specified range for a fixed variance of R1. The close correspondence between the tool and the Monte Carlo results shown in these two plots reiterates the accuracy of the tool.

The relative error between the results of the two techniques further clarifies the comparison. The error is calculated as the difference between the means and the variances of the output obtained by both the techniques taking Monte Carlo simulation results as the points of reference. Figures 4.7 and 4.8 plot this relative error in output against statistical variations in resistors R1 and R2 over the specified range of 0.1 to 1e-10. These plots are plotted on a 3-D surface and depict the trend in output error with respect to variations in the parameters of both the resistors. The error curves for the output mean in figs. 4.7 and 4.8 do not exhibit a defined trend; however, the error values are quite negligible in this case and have no impact on the tool's accuracy. The variance curves in these figures indicate that considering a single contour of the 3-D plot with a fixed variance of resistor R2, the error in the variance of the output increases with the decreasing variance of resistance R1 over the range of values from 0.1 to 1e-10. Likewise, the contour in the other direction that is obtained for a fixed variance of R1 with R2 being varied over the entire range exhibits a non-decreasing monotonicity in the output variance error with respect to variations in

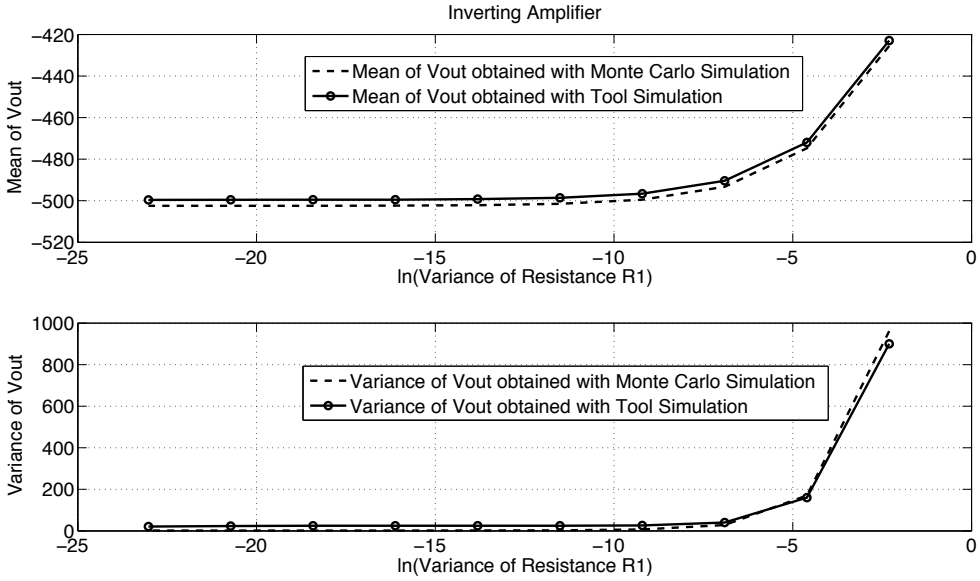

Fig. 4.3: Plot of Vout vs statistical variations in R1 for an inverting amplifier.

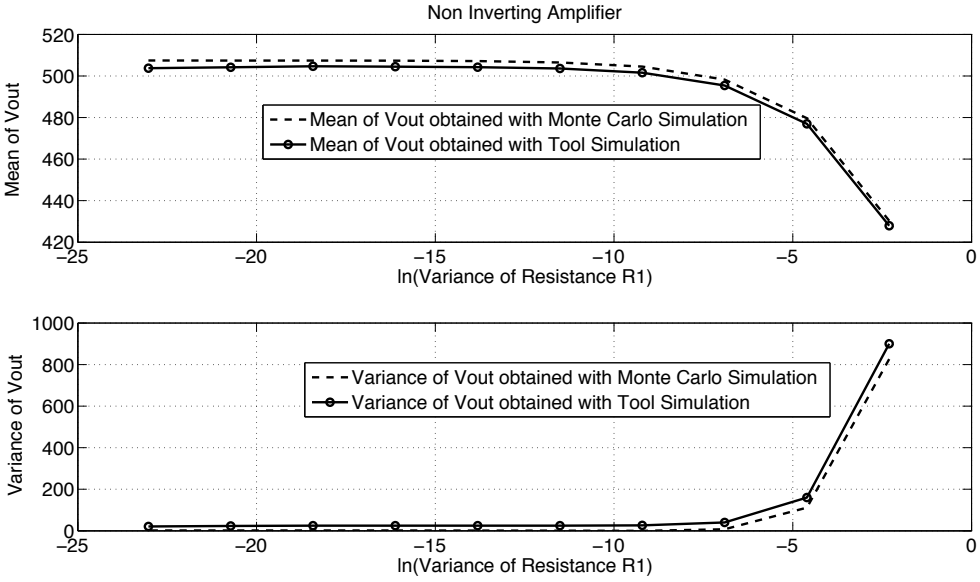

Fig. 4.4: Plot of Vout vs statistical variations in R1 for a non-inverting amplifier.

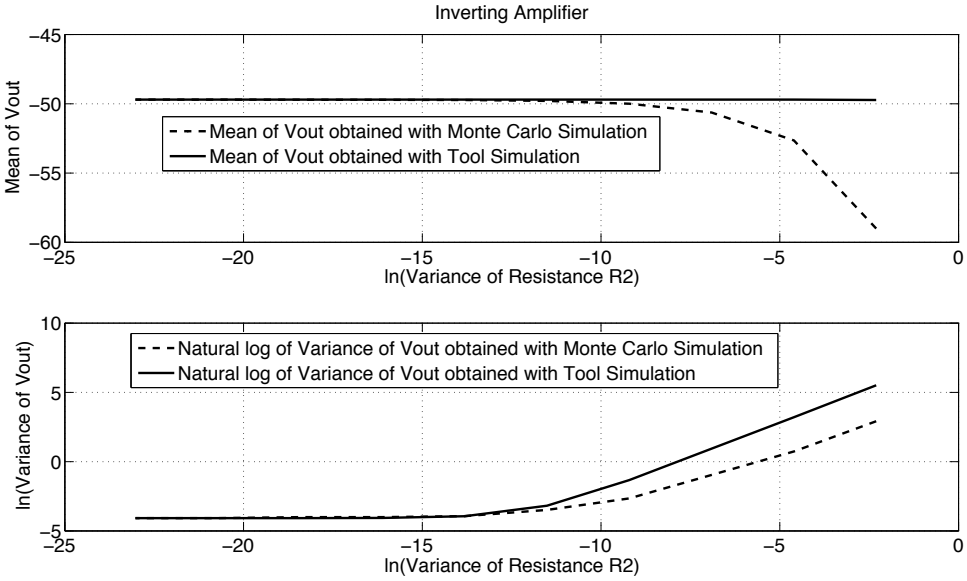

Fig. 4.5: Plot of Vout vs statistical variations in R2 for an inverting amplifier.

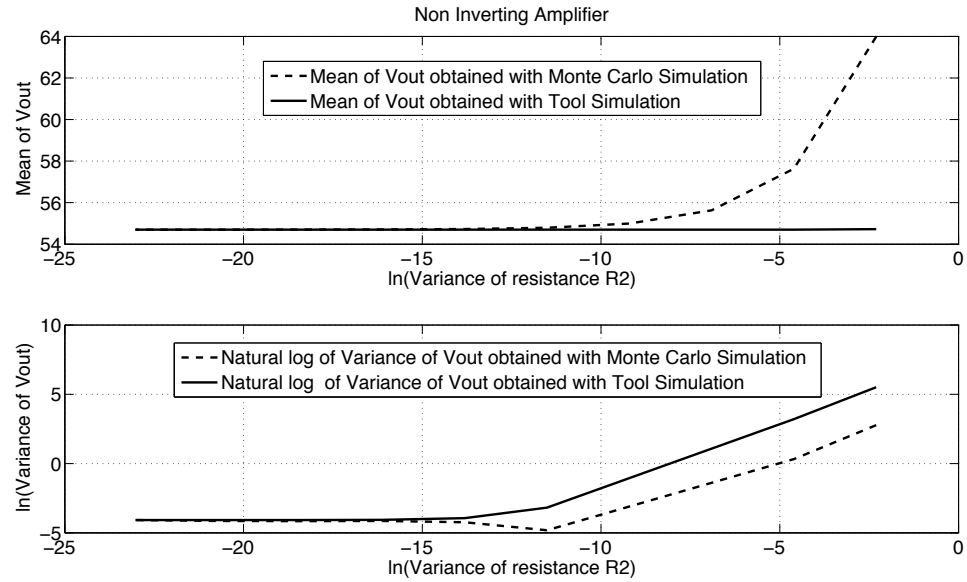

Fig. 4.6: Plot of Vout vs statistical variations in R2 for non-inverting amplifier.

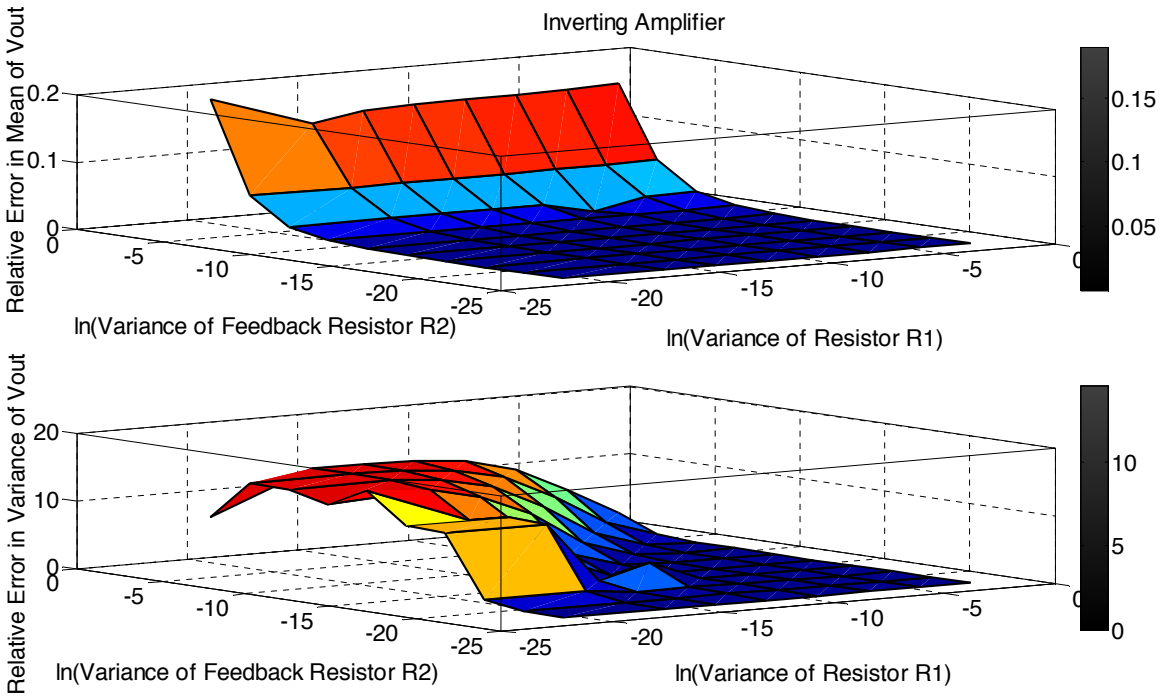

Fig. 4.7: 3-D plot of relative error in Vout for inverting amplifier against statistical variations in resistances R1 and R2.

resistor R2. The error in the variance of output decreases with decreasing variance of R2 over the given range which conforms to the expected behavior. It supports the argument that factor graph calculations are based on the Gaussian assumption of signals associated with each block and produces results comparable to the Monte Carlo simulations only for smaller variances in the circuit component parameters. However, the decreasing monotonic behavior exhibited by the output variance error curve for variations in resistor R1 for a fixed variance of R2 is unexpected and anomalous. Although it lacks concrete reasoning, the behavior is assumed to be attributed the fact that the circuit performance possesses lower sensitivity towards parametric variations in resistor R1 as opposed to feedback resistance R2. These trends in the output error are found to be consistent for both the circuit configurations.

2. Experimental Results for Backward Propagation to Calculate the Parameter Estimates: In order to estimate the unknown input parameters by propagating the known output statistics in the backward direction, output results obtained by forward propagation

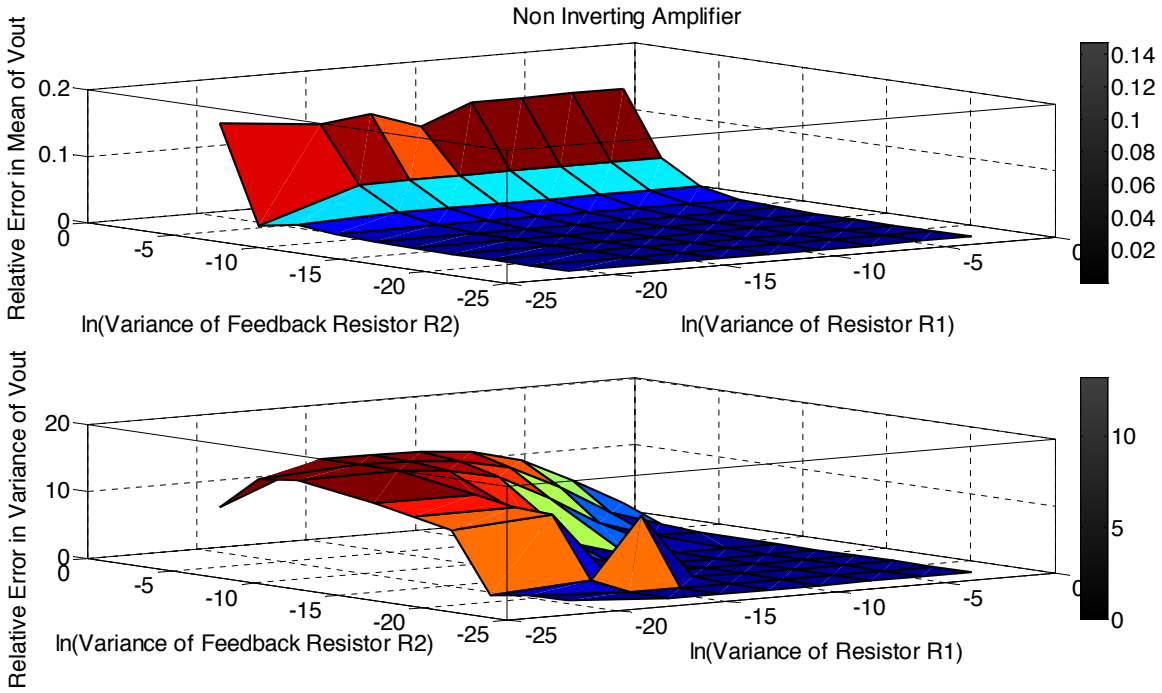

Fig. 4.8: 3-D plot of relative error in Vout for non-inverting amplifier against statistical variations in resistances R1 and R2.

of messages in the graph are used as the starting points. In this analysis, firstly output statistics for known variations in the variance of resistances R1 and R2 over the range of values 0.1 to 1e-10 are gathered. Then, the output obtained for each of the variances in the range is used to initiate backward propagation of messages in the graph to arrive at an estimate of the variance of resistors R1 and R2. The accuracy of the tool is measured in terms of the error between the estimated parameter variance obtained by the backward propagation technique and the known parameter variance used for the forward propagation. The smaller is the error between these two values, the greater is the efficiency of the tool in predicting unknown parameters accurately.

Figures 4.9 and 4.10 demonstrate the results of the backward analysis for inverting and non-inverting amplifiers, respectively. The dotted curves in these figures represent known parametric distributions of the resistances R1 and R2 that lead to the given statistics of Vout by forward message propagation in the graph as plotted on the X-axis. The solid curves, on the other hand, show the values of estimated variances of the resistors R1 and

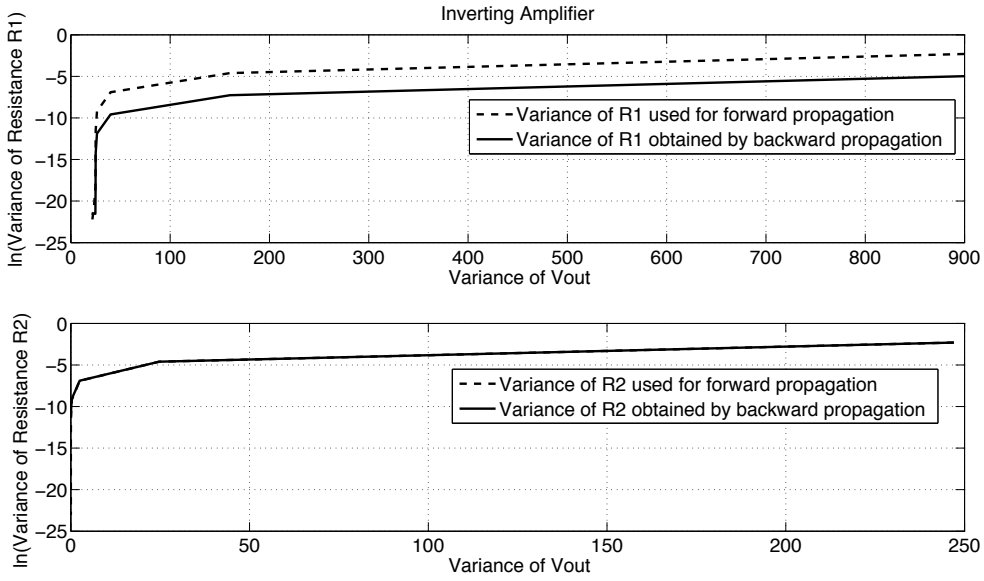

Fig. 4.9: Plot showing estimated and expected variances of resistance R1 and feedback resistor R2 of an inverting amplifier for known statistics of Vout.

R2 obtained by propagating the calculated output of the circuit along the backward edges in the graph. The results indicate that the tool could be used to determine the impact of a given output voltage constraint on the statistics of input parameters.

4.1.2 Instrumentation Amplifier

The circuit diagram of an instrumentation amplifier is shown in fig. 4.11. The forward and backward analysis runs are also performed for an instrumentation amplifier by varying the variance of its resistors R1 and R2 over a range of 0.1 to 1e-10 similar to the inverting and non-inverting configurations of the amplifier. The open-loop gain of all the three amplifiers in this circuit is fixed at a value of 100K.

1. Plots Indicating the Results Obtained by Forward Analysis of the Circuit: Figures 4.12 and 4.13 depict the values of output statistics obtained for the variations in the variance of resistances R1 and R2, respectively. The plots indicate that the simulation results obtained from the tool match approximately with those obtained from Monte Carlo simulation.

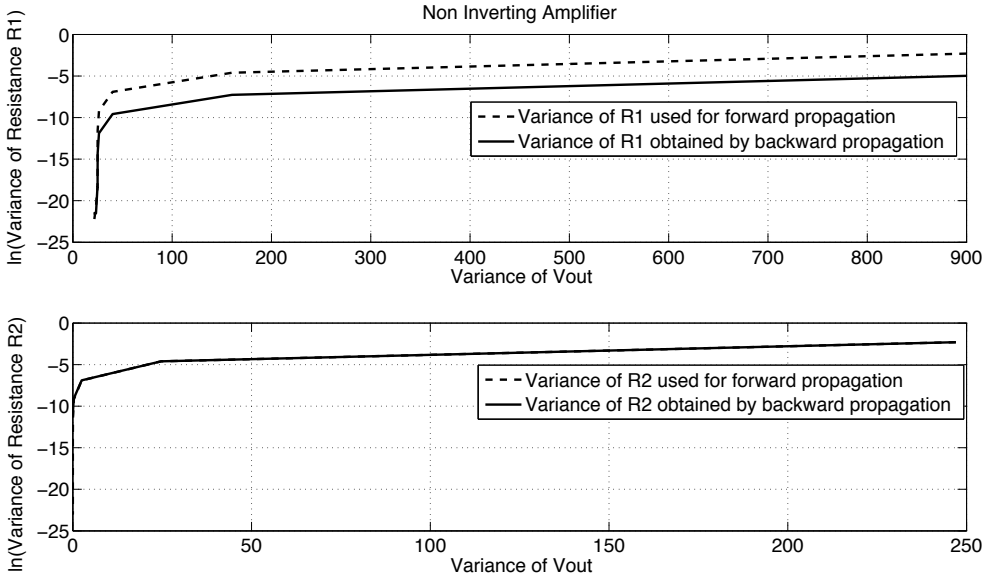

Fig. 4.10: Plot showing estimated and expected variances of resistance R1 and feedback resistor R2 of a non-inverting amplifier for known statistics of Vout.

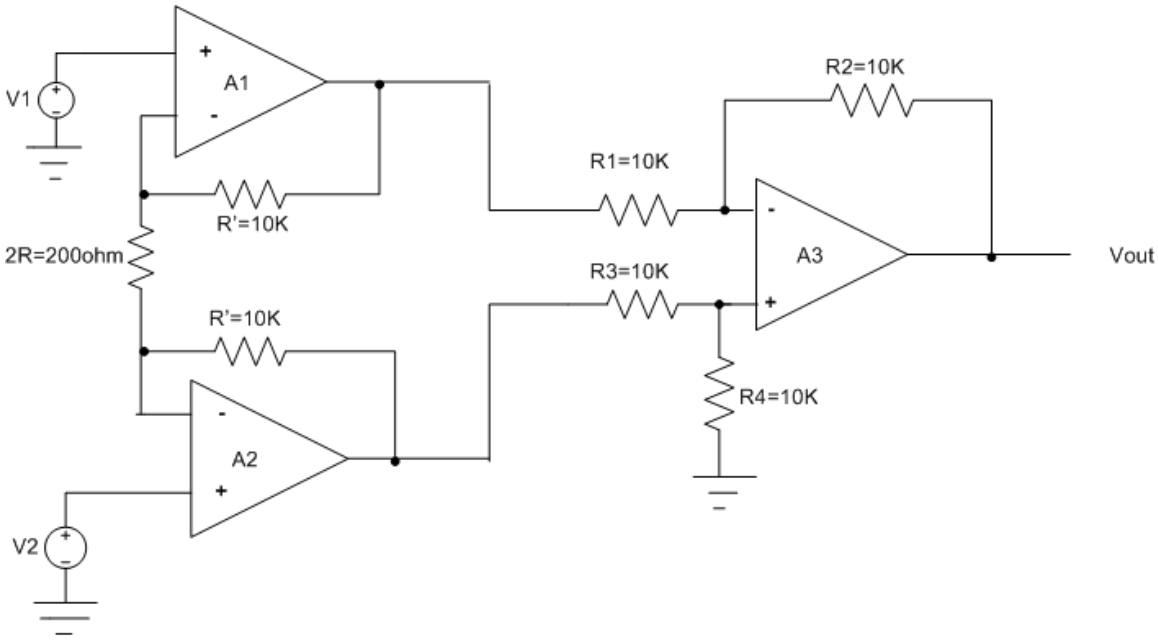

Fig. 4.11: Circuit diagram of an instrumentation amplifier.

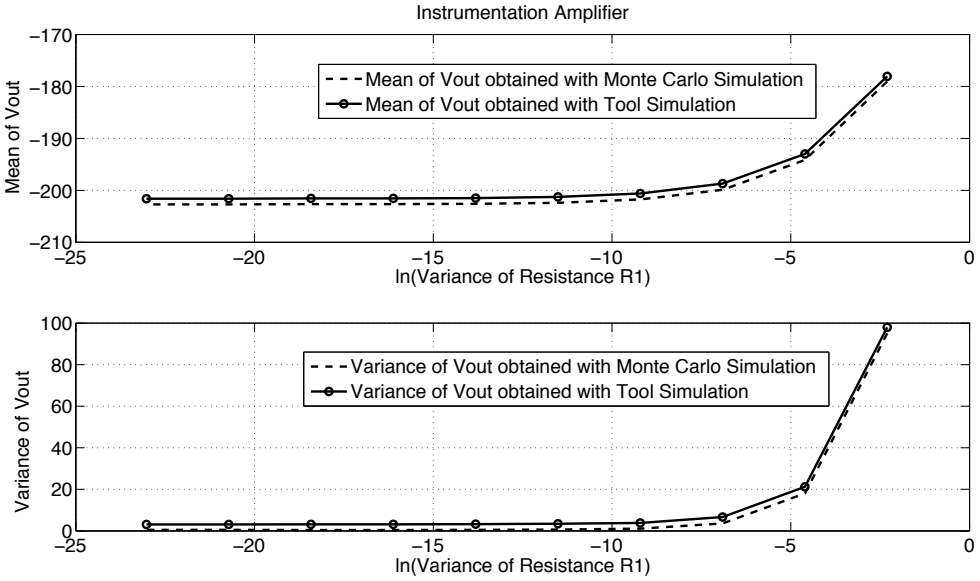

Fig. 4.12: Plot of Vout vs statistical variations in R1 for an instrumentation amplifier.

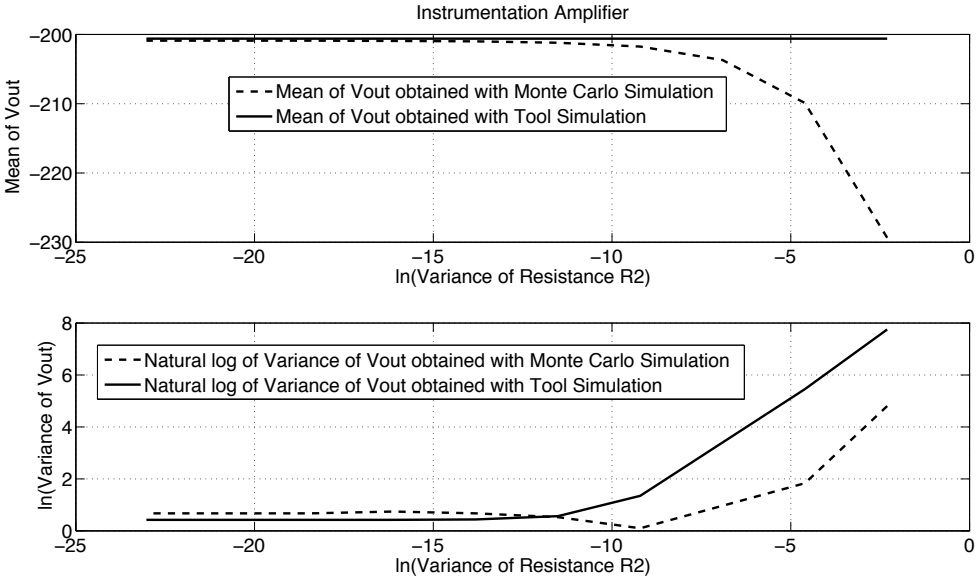

Fig. 4.13: Plot of Vout vs statistical variations in R2 for an instrumentation amplifier.

Figure 4.14 shows the 3-D plot of the relative error between the output values obtained by both the techniques against statistical variations in resistors R1 and R2. The error plots for the instrumentation amplifier indicate similar trends as observed in the circuits of inverting and non-inverting configurations of the amplifier considering. The error in the output mean is negligible over the entire range of parameter variation. The error in the variance of the output exhibits opposite trends for the variations in resistors R1 and R2, similar to the other amplifier circuits. These trends are observed by considering the single contours of the plot at fixed variance of resistors R1 and R2 when varying the variance of the other resistance over the entire range.

2. Simulation Results of Backward Analysis: The backward simulation runs are performed on the circuit to obtain parameter estimates for a known set of values of output statistics. The plots of the parameter variances obtained by propagating the known output in the backward direction in the graph for the instrumentation amplifier are shown in fig. 4.15. These plots also depict a close agreement between the estimated and expected values of the variance of resistors R1 and R2.

4.1.3 Conclusions Derived from Observed Results

An analytic comparison of all the results discussed in the previous subsections establish the fact that the circuit performance is more sensitive to the statistical variations in the feedback resistance R2 as compared to the variations in resistance R1. The plots in figs. 4.5, 4.6, and 4.13 showing the values of output mean obtained by both the techniques by varying R2's variance at a fixed value of R1 for all the three amplifier circuits indicate some amount of discrepancy between the output mean values obtained by the tool and Monte Carlo technique for larger variances of R2. The bias effect introduced by the Monte Carlo simulation which is solely a property of Monte Carlo technique is supposed to be responsible for such behavior. It thus establishes the fact of tool producing more correct results than the reference technique for larger parametric variations in feedback resistor. The output results obtained by forward analysis of the circuits conform the efficiency of the tool in yield estimation for a given set of inputs. The backward analysis plots for all the

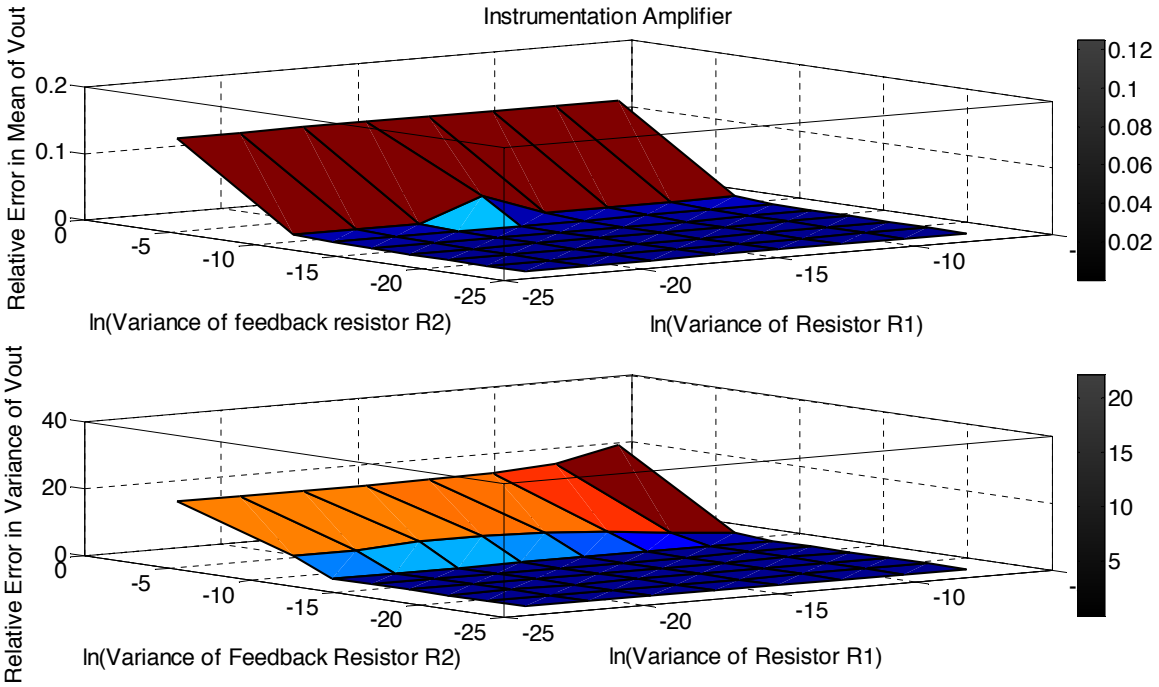

Fig. 4.14: 3-D plot of relative error in Vout for an instrumentation amplifier against statistical variations in resistances R1 and R2.

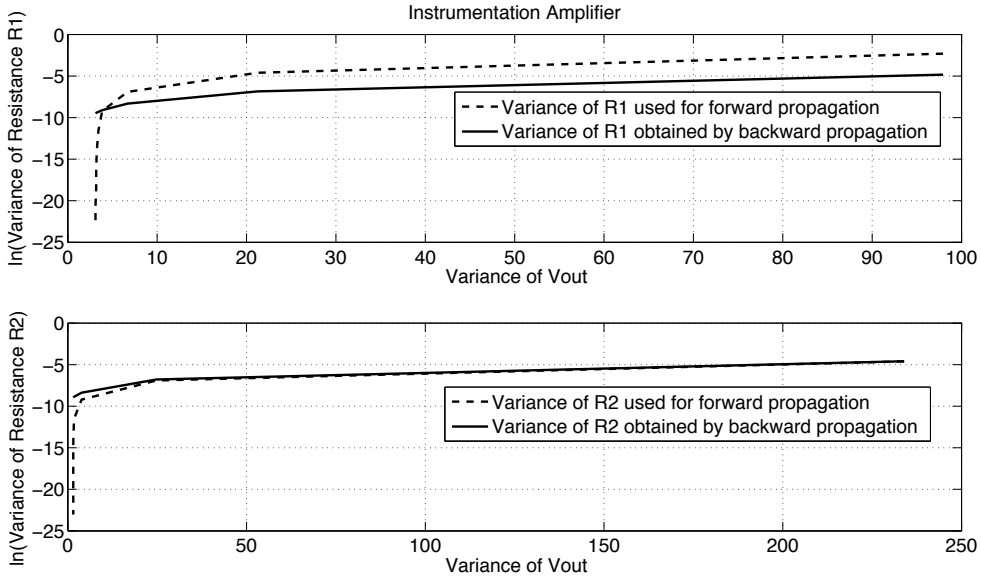

Fig. 4.15: Plot showing estimated and expected variances of resistance R1 and feedback resistor R2 of an instrumentation amplifier for known statistics of Vout.

three circuits indicate that the tool could also be used to determine the impact of a given constraint on the statistics of the output voltage on the statistics of circuit parameters. Similar results were obtained for all the three circuits with different values of resistors, thus changing the closed-loop gain. This indicates that for all the three circuits examined, the results obtained are not a function of the circuit parameters used.

4.1.4 MOSFET

In order to display the capability of the tool to design and simulate circuits with dependent sources, a small-signal equivalent circuit of MOSFET is analyzed by the tool to obtain yield and parameter estimates. Both types of analysis runs, forward and backward, are performed on the circuit by varying the variances of its resistances Rg and Rout as indicated in the circuit diagram of the MOSFET in the fig. 4.16.

1. Forward Simulation Results for MOSFET: The output statistics for the circuit are gathered by varying the variances of resistors Rg and Rout over the range 0.1 to 1e-10 through forward message propagation in its equivalent factor graph. The results of this analysis have been plotted in figs. 4.17 and 4.18. The relative errors between the output values obtained by the factor graph and the Monte Carlo techniques are plotted in fig. 4.19. An analytic study of these 3-D curves similar to previous example circuits to observe defined trends in the output values indicate that the error in the mean of Vout increases with the decreasing variance of resistor Rg while the variance error shows a decrease in value with decreasing variance of the resistance. The error plots of mean and variance against statistical

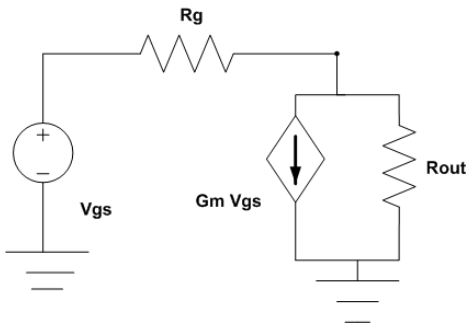

Fig. 4.16: Circuit diagram of MOSFET.

Table 4.3: Average execution time obtained from tool and Monte Carlo analysis.

Circuit	Average Run Time of Monte Carlo Technique	Average Run Time of Entire Tool Chain	Average Run Time of Factor Graph Simulator Simulator Component of Tool	No. of Factor Graph Nodes
Inverting Amplifier	14 minutes	17 seconds	<1 second	15
Non Inverting Amplifier	14 minutes	17 seconds	<1 second	15
Instrumentation Amplifier	14 minutes	18 seconds	1 second	45
MOSFET	14 minutes	17 seconds	<1 second	17

variations in the output resistor Rout, however, show exact opposite trends. The error in output mean decreases while the error in output variance increases with the decreasing variance of Rout. The values of errors in mean are very small of the order of 1e-1 and can be neglected as per the tolerance standards of the user.

2. Backward Simulation Results for MOSFET: The backward message propagation is performed along the edges of the factor graph equivalent of the MOSFET, taking the output statistics gathered from the forward propagation as starting inputs. This results in the estimation of parameter variances for resistors R1 and R2 for known yield constraints. The plot obtained from this analysis is shown in fig. 4.20. The results indicate that the tool's prediction of the unknown variances of the resistors for known output statistics match closely with the expected values.

4.2 Timing Results

In order to evaluate the efficiency of the tool in terms of total execution time, timing results obtained from Monte Carlo technique and the entire tool chain were recorded for all four circuit configurations. Table 4.3 lists the timing results for all the circuits analyzed by the tool. The time required by the factor graph simulator component of the tool chain to perform statistical analysis of the resultant factor graph model was also collected separately to establish fair grounds of comparison between Monte Carlo and factor graph analysis. These results give a clear indication of the superiority of the tool in computing desired yield

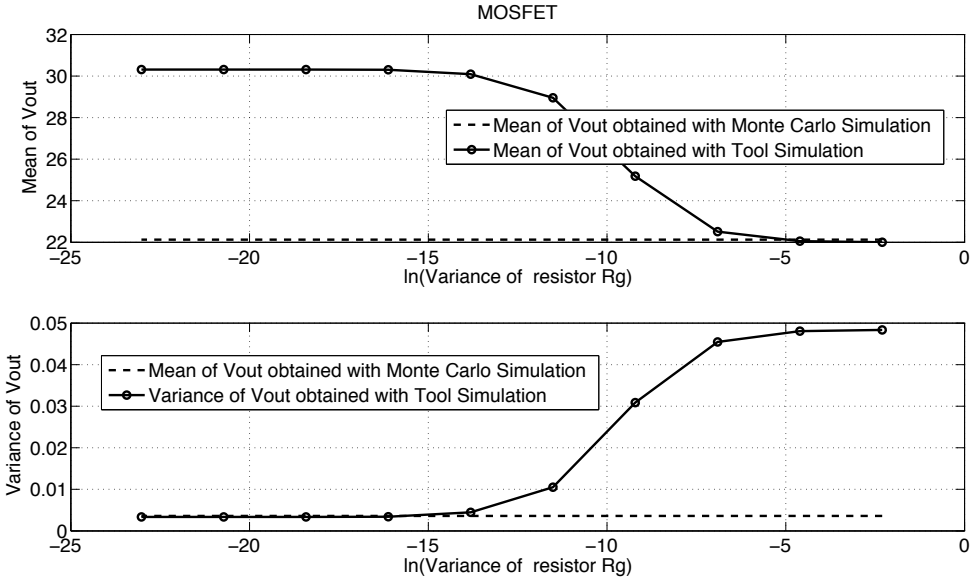

Fig. 4.17: Plot of Vout vs statistical variations in input resistance Rg for MOSFET.

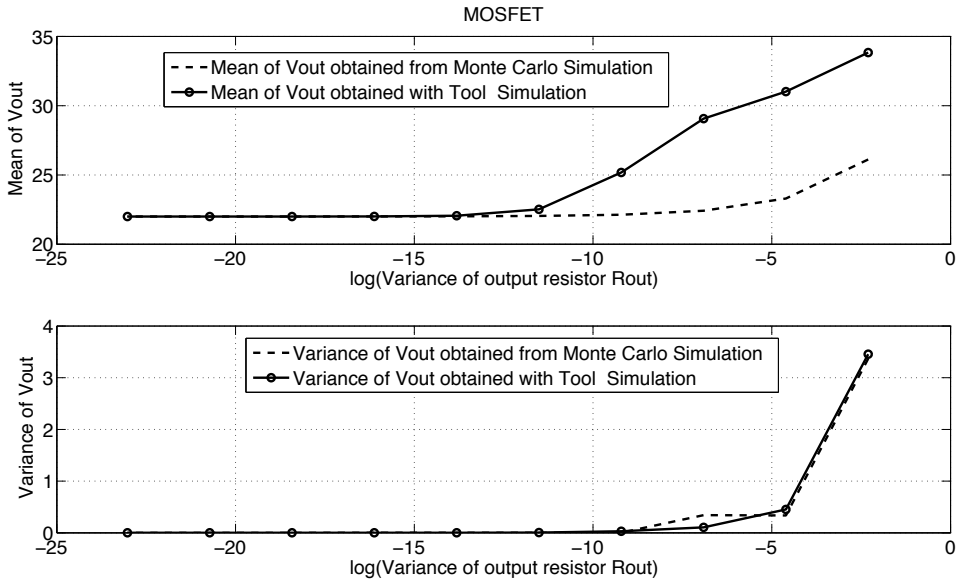

Fig. 4.18: Plot of Vout vs statistical variations in output resistance Rout for MOSFET.

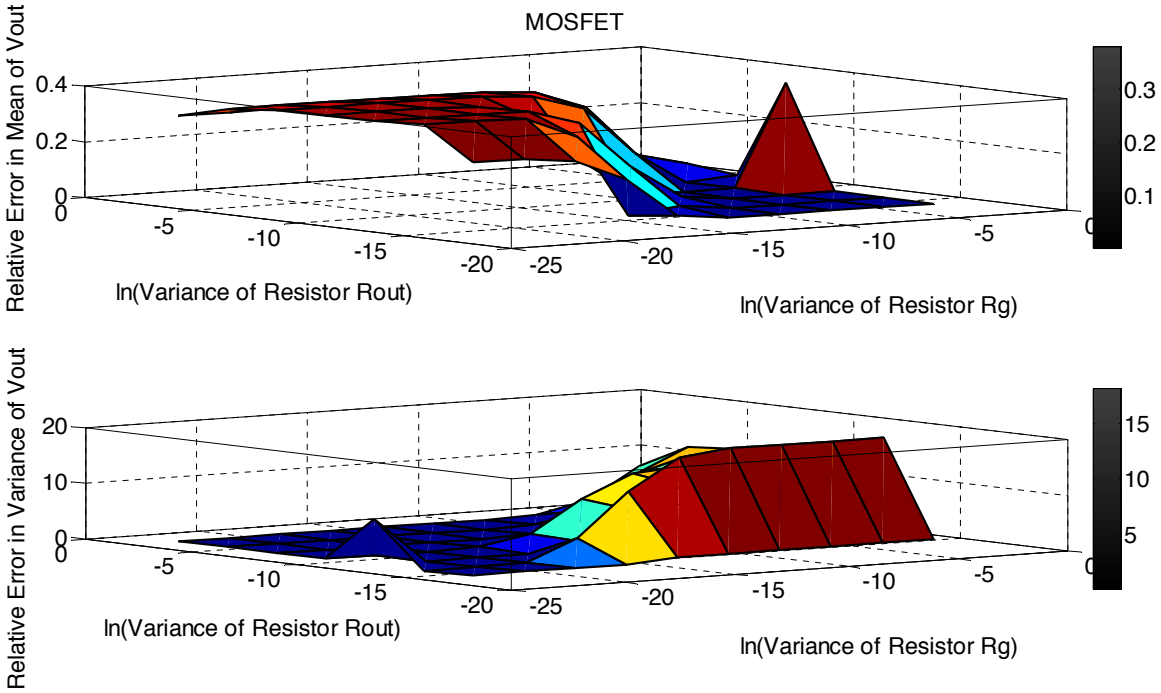

Fig. 4.19: 3-D plot of relative error in Vout for MOSFET against statistical variations in resistances Rg and Rout.

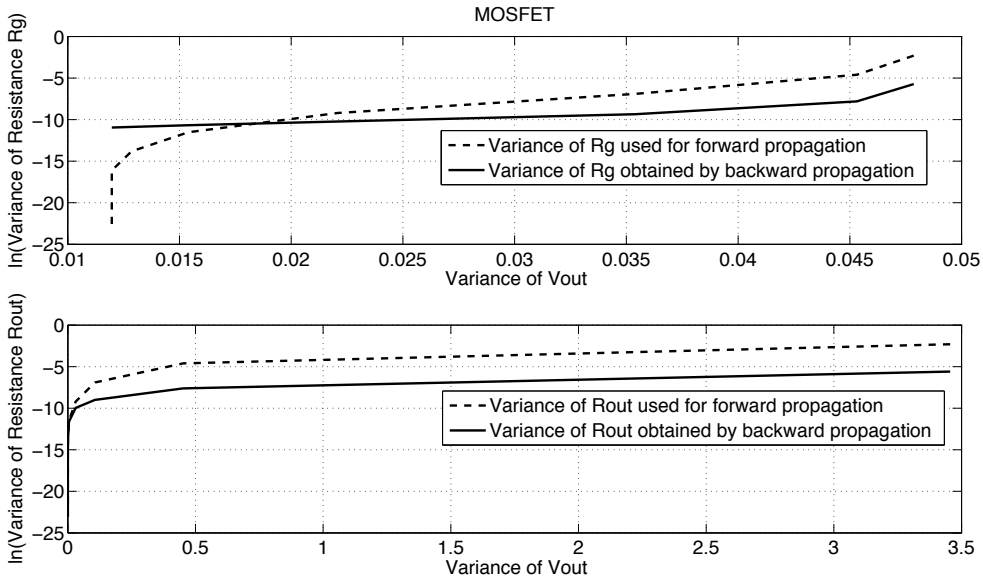

Fig. 4.20: Plot showing estimated and expected variances of input resistance Rg and output resistor Rout for MOSFET.

over Monte Carlo technique.

Chapter 5

Conclusion and Future Work

This thesis examines an approach of using factor graphs and message passing to implement statistical analysis of linear analog circuits. An approach for translating a linear analog circuit into a factor graph has been presented. The factor graphs have been examined as a means for implementing statistical analysis of circuit behavior in the presence of parameter variations. Several simulation and analysis results have been presented, comparing the factor graph approach against Monte Carlo simulation for a small number of circuits, which demonstrate the level of effectiveness of the factor graph technique in terms of a small observed error between the tool and Monte Carlo results. The factor graph approach has been analyzed both from the perspective of determining circuit output statistics, as well as a tool for determining the impact of yield constraints imposed on circuit outputs as it affects circuit parameters. The analysis results do reflect certain anomalies in the behavior of the tool with respect to Monte Carlo simulations. There exist accuracy issues at certain observation points within the specified variance range of the resistors in the all the four circuit configurations. These unexpected behaviors have been just reported in this thesis without any explanation. They probably open a new dimension of research towards mathematically analyzing their causes and effects in this analysis. Despite of the inaccuracies exhibited by the tool simulations, it could be used in the early stages of the design process to evaluate different possible design configurations for the same circuit. Moreover, the plots showing output results obtained from the tool and Monte Carlo technique in Chapter 4 indicate that the error in the output results from both the techniques is optimistic for majority of observation points. The tool produces larger variances in output as compared to Monte Carlo simulations at approximately 66.75% of the total observation points which include variances of the two resistors over the specified range of 0.1 to 1e-10 for all the circuit

configurations. Considering this fact of Monte Carlo technique producing improved values of output variances as compared to the tool, it is possible to rely on the tool's results to provide a bound on the worst case performance of the circuit.

Nevertheless, the greatest advantage of our technique lies in its ability to produce yield analysis results at a much quicker rate than the standard Monte Carlo technique. The execution time of the analysis in our approach is reduced by several orders of magnitude. The tool is approximately fifty times faster than Monte Carlo technique in performing yield analysis of analog circuits. This claim follows from the timing results indicated in Chapter 4. These results demonstrate a linear increase in time shown by factor graph technique in simulating instrumentation amplifier circuit that contains thrice as many factor graph nodes as other circuits. Our technique is particularly useful for simulating large scale circuits consisting of many circuit components in which case the run time growth of Monte Carlo technique is polynomial and in certain cases, even exponential. However, the factor graph approach follows a linear rise in the execution time with the number of factor graph nodes in the equivalent factor graph model. Thus, it is possible to perform faster analysis of analog circuits using factor graph approach.

This tool possesses scope for enhancements in several dimensions. The possible areas of the tool that hold potential for extension have been listed below.

- Although, the tool's scope is currently limited to perform steady-state analysis of analog circuits, it possesses the potential for enhancements for modeling frequency dependent components like capacitors and inductors to handle transient analysis. The tool could be extended to derive generalized voltage and current sub graph models similar to a resistor representing electrical laws of charge and flux obeyed by these components.

- The tool is able to handle nine basic kinds of junctions discussed in Chapter 3. It is possible to explore more types of junctions based upon their number of incoming and outgoing branches, and devise rules of translation for them using the existing equivalent factor graph block combinations for the nine primitive types of junctions.

The translation rules for the junctions discussed in Chapter 3 form the basis for deriving equivalent factor graph models for other kinds of junctions. There lies scope in extending the capability of the translator interpreter to code the translation rules for converting various other kinds of junctions into factor graph domain. This would widen the scope of the analog circuits being handled by the tool.

- The tool possesses the ability to model digital and mixed signal circuits using digital circuits components available in the analog circuit modeling environment. However, it still lacks the ability to convert digital circuits into their equivalent factor graph blocks. The translator interpreter could be extended to accommodate the translation rules for the digital circuit components, incorporating the ability to analyze digital and mixed signal circuits by the tool.

The proposed methodology has explored a new dimension of the application of factor graphs as a modeling tool and will continue to be a topic of interest in future.

References

[1] A. Marshall and M. Thornton, *Mismatch And Noise In Modern Ic Processes*. Dallas, TX: Morgan & Claypool Publishers, 2009.

[2] M. W. Tian and C.-J. R. Shi, "Worst case tolerance analysis of linear analog circuits using sensitivity bands," *IEEE Transactions on Circuits and Systems*, vol. 47, p. 8, 2000.

[3] F. El-Turky and E. E. Perry, "BLADES: An artificial intelligence approach to analog circuit design," *IEEE Transactions on Computer-aided Design.*, vol. 8, p. 13, 1989.

[4] A. Graupner, W. Schwarz, and R. Schuffny, "Statistical analysis of analog structures through variance calculation," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions*, vol. 49, no. 8, pp. 1071–1078, Aug. 2002.

[5] R. Rodriguez-Macias and A. Rodriguez-Vazquez, "A technique for fast ac statistical analysis of analog circuits," *Electronics, Circuits and Systems, 1998 IEEE International Conference*, vol. 2, pp. 81–84, 1998.

[6] S. Ali, P. Wilson, and A. Brown, "Yield predictive model characterization in analog circuit design," *Integrated Circuits, 2007. International Symposium*, pp. 289–292, Sept. 2007.

[7] G. Gielen, P. Wambacq, and W. Sansen, "Symbolic analysis methods and applications for analog circuits: a tutorial overview," *Proceedings of the IEEE*, vol. 82, no. 2, pp. 287–304, Feb. 1994.

[8] H.-A. Loeliger, "An introduction to factor graphs," *Signal Processing Magazine, IEEE*, vol. 21, no. 1, pp. 28–41, Jan. 2004.

[9] B. J. Frey, F. R. Kschischang, H.-A. Loeliger, and N. Wiberg, "Factor graphs and algorithms," [http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.27.4388], 1998.

[10] H.-A. Loeliger, J. Dauwels, J. Hu, S. Korl, L. Ping, and F. Kschischang, "The factor graph approach to model-based signal processing," *Proceedings of the IEEE*, vol. 95, no. 6, pp. 1295–1322, June 2007.

[11] F. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, pp. 498–519, 2001.

[12] H.-A. Loeliger, "Calibration of analog-to-digital converters with low-precision components," in *The 2007 Analog Decoding Workshop*, 2007.

[13] Vanderbilt University, "GME User's Manual," 2004.

[14] "Gme:generic modeling environment," [http://www.isis.vanderbilt.edu/Projects/gme], 2008.

[15] H.-A. Loeliger, J. Hu, S. Korl, Q. Guo, and L. Ping, "Gaussian message passing on linear models: An update," *Turbo-Coding-2006*, pp. 1–7, 2006.

[16] G. Colavolpe and G. Germi, "On the application of factor graphs and the sum product algorithm to isi channels," *IEEE Transactions on Communications*, vol. 53, p. 8, 2005.

[17] "Bcjr algorithm," [http://en.wikipedia.org/wiki/BCJR_algorithm], 2009.

[18] S. Korl, H. Loeliger, and A. Lindgren, "Ar model parameter estimation: from factor graphs to algorithms," in *Acoustics, Speech, and Signal Processing, 2004. Proceedings. IEEE International Conference*, vol. 5, pp. V–509–12, May 2004.

[19] "Autoregressive model," [http://en.wikipedia.org/wiki/Autoregressive_model], 2009.

[20] A. Eckford, "The factor graph em algorithm: applications for ldpc codes," in *2005 IEEE 6th Workshop on Signal Processing Advances in Wireless Communications*, pp. 910–914, 2005.

[21] P. W. Tuinenga, *SPICE: A Guide to Circuit Simulation & Analysis Using PSpice*. Englewood Cliffs, NJ: Prentice Hall, Inc., 1992.

[22] "ecircuit center," [http://www.ecircuitcenter.com/Circuits/opmodel1/opmodel1.htm], 2003.

[23] "Pspice tutorials," [http://www.uta.edu/ee/hw/pspice/], 2009.

[24] J. Keown, *MicroSim PSpice and Circuit Analysis*. Columbus, OH: Prentice Hall, Inc., 1998.

[25] A. S. Sedra and K. C. Smith, *Microelectronic Circuits*. New York: Oxford University Press, Inc., 2004.

Appendices

Appendix A

Equations Guiding Forward and Backward Message Propagation Through Each Factor Graph Node

Sections A.1 and A.2 discuss the equations transforming the signals propagating through each node in a factor graph. All messages are assumed to be Gaussian mixtures with a mean, variance, weight, and density attribute associated with them. The notations used for representing the mean, variance, weight, and density attributes of a Gaussian mixture are m, V, W, and p, respectively.

A.1 Equations for Forward Message Propagation Along a Factor Graph

This section illustrates the equations guiding the forward message propagation along the edges of the graph. Below is a list of all the factor graph nodes and their associated equations.

- Adder: Z=X+Y

$$Z.p = X.p * Y.p \tag{A.1}$$

$$Z.m = X.m + Y.m \tag{A.2}$$

$$Z.V = X.V + Y.V \tag{A.3}$$

$$Z.W = 1/Z.V \tag{A.4}$$

- Coefficient: Z=X*A*(1+eps)

$$Z.p = X.p * eps.p \tag{A.5}$$

$$Z.m = A * X.m \tag{A.6}$$

$$Z.V = (A * X.m)^2 * eps.V + (A^2) * X.V \tag{A.7}$$

$$Z.W = 1/Z.V \tag{A.8}$$

- Scalar Multiplier: Y=A*X

$$Y.p = X.p \tag{A.9}$$

$$Y.m = A * X.m \tag{A.10}$$

$$Y.V = (A^2) * X.V \tag{A.11}$$

$$Y.W = 1/Y.V \tag{A.12}$$

- Equality: Z=Y=X

$$Z.p = X.p * Y.p \tag{A.13}$$

$$Z.W = X.W + Y.W \tag{A.14}$$

$$Z.m = (X.W * X.m + Y.W * Y.m)/Z.W \tag{A.15}$$

$$Z.V = 1/Z.W \tag{A.16}$$

A.2 Equations for Backward Message Propagation Along a Factor Graph

This section presents the equations associated with each factor graph node that guide the transformation of the signals along the backward edges of the graph.

- Adder: Z=X+Y

  1. Outgoing Message on X:

$$X.p = Y.p * Z.p \tag{A.17}$$

$$X.m = Z.m - Y.m \tag{A.18}$$

$$X.V = Z.V + Y.V \tag{A.19}$$

$$X.W = 1/X.V \tag{A.20}$$

2. Outgoing Message on Y:

$$Y.p = X.p * Z.p \tag{A.21}$$

$$Y.m = Z.m - X.m \tag{A.22}$$

$$Y.V = Z.V + X.V \tag{A.23}$$

$$Y.W = 1/Y.V \tag{A.24}$$

- Coefficient: Z=X*A*(1+eps)

1. Outgoing message on X:

$$X.p = Z.p * eps.p \tag{A.25}$$

$$X.m = Z.m/A \tag{A.26}$$

$$X.V = Z.V/(A^2) + eps.V * (Z.m/A)^2 \tag{A.27}$$

$$X.W = 1/X.V \tag{A.28}$$

2. Outgoing message on eps:

$$eps.p = Z.p * X.p \tag{A.29}$$

$$eps.m = 0 \tag{A.30}$$

$$eps.V = Z.V/((A * X.m)^2) + X.V * ((Z.m/(A * X.m^2))^2) \tag{A.31}$$

$$eps.W = 1/eps.V \tag{A.32}$$

- Scalar Multiplier: Y=A*X

$$X.p = Y.p \tag{A.33}$$

$$X.W = Y.W * A^2 \tag{A.34}$$

$$X.m = A * Y.W * Y.m / X.W \tag{A.35}$$

$$X.V = 1/X.W \tag{A.36}$$

- Equality: Z=Y=X

  1. Outgoing Message on X:

$$X.p = Z.p * Y.p \tag{A.37}$$

$$X.W = Z.W + Y.W \tag{A.38}$$

$$X.m = (Z.W * Z.m + Y.W * Y.m)/X.W \tag{A.39}$$

$$X.V = 1/X.W \tag{A.40}$$

  2. Outgoing Message on Y:

$$Y.p = Z.p * X.p \tag{A.41}$$

$$Y.W = Z.W + X.W \tag{A.42}$$

$$Y.m = (Z.W * Z.m + X.W * X.m)/Y.W \tag{A.43}$$

$$Y.V = 1/Y.W \tag{A.44}$$

Appendix B

Sample of the C++ Simulation File Generated by the Factor Graph Interpreter

```
#include "Message.h"
#include "component.h"
#include <iostream>
#include <list>
#include <fstream>
using namespace std;
int main(int argc, char* argv[])
{
bool done=false;
ofstream output_txt;
list <component*>allobjs;
component *comp=new component();
bool oport0[]=0;
double mean0[]=0;
double variance0[]=0.1;
double weight0[]=10;
double wght_densities0[]=1;
bool oport1[]=0;
double mean1[]=0;
double variance1[]=1e-4;
double weight1[]=1e+4;
double wght_densities1[]=1;
```

bool oport2[]=0;

double mean2[]=5;

double variance2[]=1e-8;

double weight2[]=1e+8;

double wght_densities2[]=1;

bool oport3[]=0;

double mean3[]=0.042325;

double variance3[]=8.85062e-006;

double weight3[]=112986;

double wght_densities3[]=1;

bool oport4[]=0;

double mean4[]=0.000E+00;

double variance4[]=1e-18;

double weight4[]=1e+18;

double wght_densities4[]=1;

bool oport5[]=0;

double mean5[]=0.042325;

double variance5[]=8.85062e-006;

double weight5[]=112986;

double wght_densities5[]=1;

bool iport6[]=0,0;

bool oport6[]=0;

double mean6[]=0;

double variance6[]=0.001;

double weight6[]=1000;

double wght_densities6[]=1;

bool iport7[]=0,0;

bool oport7[]=0;

```
double mean7[]=0;
double variance7[]=0.001;
double weight7[]=1000;
double wght_densities7[]=1;
bool iport8[]=1,0;
bool oport8[]=0;
double mean8[]=0;
double variance8[]=0.001;
double weight8[]=1000;
double wght_densities8[]=1;
bool iport9[]=0;
bool oport9[]=0,0;
double mean9[]=0;
double variance9[]=0.001;
double weight9[]=1000;
double wght_densities9[]=1;
bool iport10[]=1,0;
bool oport10[]=0;
double mean10[]=0;
double variance10[]=0.001;
double weight10[]=1000;
double wght_densities10[]=1;
bool iport11[]=0,1;
bool oport11[]=0;
double mean11[]=0;
double variance11[]=0.001;
double weight11[]=1000;
double wght_densities11[]=1;
```

```
bool iport12[]=0;
bool oport12[]=0;
double mean12[]=0;
double variance12[]=0.001;
double weight12[]=1000;
double wght_densities12[]=1;
bool iport13[]=0,0;
bool oport13[]=0;
double mean13[]=0;
double variance13[]=0.001;
double weight13[]=1000;
double wght_densities13[]=1;
bool iport14[]=0;
double mean14[]=1;
double variance14[]=1000;
double weight14[]=0.001;
double wght_densities14[]=1;
errorsource *es2=new errorsource("ES1", 0, 1, mean0, variance0, weight0, wght_densities0,
oport0);
allobjs.push_back(es2);
errorsource *es4=new errorsource("ES2", 0, 1, mean1, variance1, weight1, wght_densities1,
oport1);
allobjs.push_back(es4);
input *i7=new input("Vsig1", 0, 1, mean2, variance2, weight2, wght_densities2, oport2);
allobjs.push_back(i7);
input *i11=new input("I2", 0, 1, mean3, variance3, weight3, wght_densities3, oport3);
allobjs.push_back(i11);
input *i13=new input("V3", 0, 1, mean4, variance4, weight4, wght_densities4, oport4);
```

```
allobjs.push_back(i13);
input *i14=new input("I1", 0, 1, mean5, variance5, weight5, wght_densities5, oport5);
allobjs.push_back(i14);
coefficient *c1=new coefficient("R1", 2, 1, mean6, variance6, weight6, wght_densities6,
iport6,oport6, 100, 0, 1);
allobjs.push_back(c1);
coefficient *c3=new coefficient("R2", 2, 1, mean7, variance7, weight7, wght_densities7,
iport7, oport7, 1000, 0, 1);
allobjs.push_back(c3);
adder *a15=new adder("adder", 2, 1, mean8, variance8, weight8, wght_densities8,
iport8, oport8);
allobjs.push_back(a15);
equality *e10=new equality("equality_junction1", 1, 2, mean9, variance9, weight9,
wght_densities9, iport9, oport9);
allobjs.push_back(e10);
adder *a6=new adder("adder_ampr_Amplifier", 2, 1, mean10, variance10, weight10,
wght_densities10, iport10, oport10);
allobjs.push_back(a6);
adder *a9=new adder("adder_junction1", 2, 1, mean11, variance11, weight11,
wght_densities11, iport11, oport11);
allobjs.push_back(a9);
scalarmultiplier *m5=new scalarmultiplier("Amplifier", 1, 1, mean12, variance12,
weight12, wght_densities12, iport12, oport12, 100000);
allobjs.push_back(m5);
equality *e12=new equality("equality_junction2", 2, 1, mean13, variance13,
weight13, wght_densities13, iport13, oport13);
allobjs.push_back(e12);
output *o8=new output("Vout", 1 ,0, mean14, variance14, weight14, wght_densities14,
```

```
iport14);
allobjs.push_back(o8);
comp->CreateConnection(es2,0,c1,0);
comp->CreateConnection(es4,0,c3,0);
comp->CreateConnection(i7,0,a15,1);
comp->CreateConnection(i11,0,c3,1);
comp->CreateConnection(i13,0,a6,1);
comp->CreateConnection(i14,0,c1,1);
comp->CreateConnection(c1,0,a15,0);
comp->CreateConnection(c3,0,a9,1);
comp->CreateConnection(a15,0,e10,0);
comp->CreateConnection(e10,0,a9,0);
comp->CreateConnection(e10,1,a6,0);
comp->CreateConnection(a6,0,m5,0);
comp->CreateConnection(a9,0,e12,0);
comp->CreateConnection(m5,0,e12,1);
comp->CreateConnection(e12,0,o8,0);
list<component*>::iterator cmpit;
int i=1;
for(cmpit=allobjs.begin();cmpit!=allobjs.end();cmpit++)
{
component *cp=*cmpit;
cp->SetInitialCurrentMessages();
}
while(!done)
{
for(cmpit=allobjs.begin();cmpit!=allobjs.end();cmpit++)
{
```

```
component *cmp=*cmpit;
cmp->forward_message_propogate();
cmp->backward_message_propogate();
}
done=comp->UpdateTimeStep(allobjs);
}
output_txt.open("out.txt", std::ios_base::out);
for(cmpit=allobjs.begin();cmpit!=allobjs.end();cmpit++)
{
component *cp=*cmpit;
cp->PrintAllMessages(output_txt);
}
output_*txt.close();
out<<"completed processing";
return 0;
}
```